Lecture Notes in Artificial Intelligence 2333

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

John-Jules Ch. Meyer   Milind Tambe (Eds.)

Intelligent Agents VIII

Agent Theories, Architectures, and Languages

8th International Workshop, ATAL 2001
Seattle, WA, USA, August 1-3, 2001
Revised Papers

Springer

Series Editors

Jaime G. Carbonell,Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

John-Jules Ch. Meyer
Utrecht University, Institute of Information and Computing Sciences
Intelligent Systems Group, Padualaan 14
De Uithof, 3508 TB Utrecht, The Netherlands
E-mail: jj@cs.uu.nl

Milind Tambe
University of Southern California, Information Sciences Institute
Henry Salvatori Computer Center 232, Los Angeles, CA 90089-0781, USA
E-mail: tambe@usc.edu

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Intelligent agents VIII : agent theories, architectures, and languages ; 8th international workshop ; revised papers / ATAL 2001, Seattle, WA, USA, August 1 - 3, 2001. John-Jules Ch. Meyer ; Milind Tambe (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2333 : Lecture notes in artificial intelligence)
ISBN 3-540-43858-0

CR Subject Classification (1998): I.2.11, I.2, C.2.4, D.2, F.3

ISSN 0302-9743
ISBN 3-540-43858-0 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH, Heidelberg
Printed on acid-free paper SPIN 10869757 06/3142 5 4 3 2 1 0

Preface

This volume is the eighth in the Intelligent Agents series associated with the ATAL workshops. These workshops on "Agent Theories, Architectures, and Languages" have established themselves as a tradition, and play the role of small but internationally well-known conferences on the subject, where besides theory *per se* also integration of theory and practice is in focus. Specifically, ATAL addresses issues of theories of agency, software architectures for intelligent agents, methodologies and programming languages for realizing agents, and software tools for applying and evaluating agent-based systems.

ATAL 2001 featured two special tracks in which both the more theoretical / formal and the more practical aspects were present, viz. "Formal Theories of Negotiation", organized by Frank Dignum, and "Agents for Hand-Held, Mobile, or Embedded Devices", organized by Tim Finin. There was also an extra session on RoboCup Rescue, organized and presented by Satoshi Tadokoro and Ranjit Nair.

ATAL 2001 attracted 68 papers from over 20 countries all over the world, of which 30 were selected for presentation at the workshop and publication in this volume. We invited two outstanding speakers: Fausto Giunchiglia (Trento, Italy) and Tom Dean (Brown, USA). The following authors / papers were given an award, sponsored by Springer:

– BEST PAPER: Sadri, Toni, and Torroni, Dialogues for negotiation: agent varieties and dialogue sequences
– BEST STUDENT PAPER: Felix Brandt (paper written jointly with Gerhard Weiss): Antisocial agents and Vickrey auctions

Finally, we would like to take this opportunity to thank all the persons involved in the realization of the workshop and this book: authors, invited speakers, PC members, additional reviewers, the associate chair, special track organizers, steering committee, the audience of the ATAL 2001 workshop, the people from Springer-Verlag, and last but not least our sponsors Springer-Verlag and the ESPRIT AgentLink2 Network of Excellence.

April 2002

John-Jules Meyer
Milind Tambe

Workshop Organization

Organizing committee

John-Jules Meyer (Co-chair)	Utrecht University, The Netherlands
Milind Tambe (Co-chair)	University of Southern California, USA
David Pynadath (Associate chair)	University of Southern California, USA

Special track organisers

"Formal Theories of Negotiation" :

Frank Dignum	Utrecht University, The Netherlands

"Agents for Hand-Held, Mobile, or Embedded Devices" :

Tim Finin	University of Maryland Baltimore County, USA

Steering committee

Nick Jennings	Southampton, UK
Yves Lesperance	York, Canada
John-Jules Meyer	Utrecht, The Netherlands
Joerg Mueller	Siemens, Germany
Munindar Singh	North Carolina State, USA
Milind Tambe	USC-ISI, USA
Michael Wooldridge	Liverpool, UK

Program chair

John-Jules Meyer	Utrecht University, The Netherlands
Milind Tambe	University of Southern California, USA

Program committee

see listing on next page

Additional reviewers

Sasikanth Avancha, Jim Blythe, Harry Chen, R. Scott Cost, Esther David, Luke Hunsberger, Lalana Kagal, Vlad Korolev, Cheryl Martin, Foster McGeary, Maria Miceli, Jill Nickerson, Charlie Ortiz, Fatma Ozcan, Marek Sergot, Steven Shapiro, S. A. Terwijn, Maksim Tsvetovat, Leon van der Torre

Program committee

Chitta Baral	Arizona State Univ., USA
Suzanne Barber	Univ. of Texas at Austin, USA
Michael Beetz	Univ. of Bonn, Germany
Cristiano Castelfranchi	CNR, Rome, Italy
Lawrence Cavedon	RMIT, Australia
Phil Cohen	Oregon Graduate Inst., USA
Rosaria Conte	IP-CNR, Italy
Giuseppe De Giacomo	Univ. of Rome, Italy
Keith Decker	Univ. of Delaware, USA
Frank Dignum	Utrecht Univ., The Netherlands
Mark d'Inverno	Univ. of Westminster, UK
Alexis Drogoul	Univ. of Paris VI, France
Ed Durfee	Univ. of Michigan, USA
Jacques Ferber	Univ. of Montpellier II, France
Tim Finin	Univ. of Maryland Baltimore County
Klaus Fischer	DFKI, Germany
Michael Fisher	Univ. of Liverpool, UK
Stan Franklin	Univ. of Memphis, USA
Fausto Giunchiglia	Univ. of Trento, Italy
Piotr Gmytrasiewicz	Univ. of Texas at Arlington, USA
Barbara Grosz	Harvard Univ., USA
Henry Hexmoor	Univ. of North Dakota, USA
Wiebe van der Hoek	Utrecht Univ., The Netherlands
Marc Huber	Intelligent Reasoning Systems, USA
Nick Jennings	Univ. of Southampton, UK
Anupam Joshi	Univ. of Maryland, USA
Hirofumi Katsuno	Tokyo Denki Univ., Japan
David Kinny	Univ. of Melbourne, Australia
Sarit Kraus	Bar-Ilan Univ., Israel
Ora Lassila	Nokia Research Center, Boston, USA
Yves Lesperance	York Univ., Toronto, Canada
Alessio Lomuscio	Imperial College, London, UK
Michael Luck	Univ. of Southampton, UK
John-Jules Meyer	Utrecht Univ., The Netherlands (co-chair)
Joerg Mueller	Siemens, Germany
Hideyuki Nakashima	AIST, Japan
Simon Parsons	Univ. of Liverpool, UK
David Pynadath	USC-ISI, USA
Anand Rao	Mitchell Madison Group, UK
Luciano Serafini	Univ. of Trento, Italy
Onn Shehory	IBM Haifa Res. Labs, Israel
Carles Sierra	CSIC, Spain
Munindar Singh	North Carolina State Univ., USA
Liz Sonenberg	Univ. of Melbourne, Australia

Table of Contents

Introduction

John-Jules Ch. Meyer[1] and Milind Tambe[2]

[1] Intelligent Systems Group
Institute of Information and Computing Sciences
Utrecht University
Padualaan 14, De Uithof
P.O.Box 80.089
3508 TB Utrecht, The Netherlands.
jj@cs.uu.nl

[2] USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292.
tambe@usc.edu

Like its predecessors this volume contains papers on Agent Theories, Architectures and Languages. Since we had special sessions on “Formal Theories of Negotiation” and “Agents for Hand-Held, Mobile, or Embedded Devices”, and we, moreover, saw an increase in multi-agent aspects, such as communication and collaboration, we have divided the papers in this volume into the following eight parts:

– Agent Modelling
– Formal Specification and Verification of Agents
– Agent Architectures and Languages
– Agent Communication
– Collaborative Planning and Resource Allocation
– Trust and Safety
– Formal Theories of Negotiation
– Agents for Hand-Held, Mobile, or Embedded Devices

Agent Modelling

In this part we have collected papers dealing with the modelling of agents.

Giunchiglia et al. show how the agent-oriented software engineering methodology Tropos, which is based on knowledge level concepts, can be used in the development of an agent-based meeting scheduler system.

Gmytrasiewicz and Lisetti employ explicit emotional states and personality in the design and modelling of agents, where emotional states are viewed as the agent’s decision-making modes, and an agent’s personality is taken as consisting of the agent’s emotional states together with the specification of transitions between these.

In *Kinny’s* paper an algebraic language, called the Ψ Calculus, is presented for modelling agents employing a sense-compute-act cycle and stored plan execution. An operational semantics is provided in process-algebraic style by means of rewite rules.

In their paper *Wagner and Lesser* present the *MQ* (motivational quantities) model for local agent control of organised agents in large-scale multi-agent systems and show its use in different applications. In this model motivational quantities represent progress toward organisational goals.

Formal Specification and Verification of Agents

Here we have put papers that are concerned with specifying and verifying agents by some formal means.

In his paper *Lespérance* discusses the problem of ensuring that agents' plans are epistemically feasible in formal specifications of multi-agent systems, in particular in the language CASL (Cognitive Agent Specification Language). An account of subjective plan execution in this language is proposed that ensures that the plan can be executed by the agent based on its knowledge state.

Lomuscio and Sergot are concerned with an extension of the interpreted systems formalism by Halpern *et al.* to model correct behaviour of agents, resulting in a notion of deontic interpreted systems. These systems are axiomatised yielding the logic $KD45_n^{i-j}$, which is a stronger version of the standard deontic logic KD.

Ryan and Schobbens present in their paper a notion of refinement between agent-oriented systems using alternating-time temporal logic. The refinement relation provides a framework for defining roles in a society of interacting agents and formalising a relation of conformance between agents and roles.

Finally, in this part, *Wooldridge and Dunne* investigate the computational complexity of the agent verification problem. In particular, this complexity is related to the complexity of the task specification, i.e. how hard it is to decide whether or not an agent has succeeded.

Agent Architectures and Languages

In ths part we have collected papers dealing with both explicit achitectural issues for agents and the use of agent languages.

Kolp et al. propose architectural styles for multi-agent systems that adopt concepts from organisation theory and the strategic alliances literature. Multi-agent systems are conceived as organisations of agents that interact to achieve common goals, and a catalogue of architectural styles and agent patterns is proposed for designing these systems at both the macro- and microlevel.

In the paper of *Leite et al.* the authors present a description of the agent architecture MINERVA, designed with the intention of providing a common agent framework based on logic programming, to allow for the combination of several non-monotonic knowledge representation and reasoning mechanisms.

Next, *Machado and Bordini* show how agents written in the programming language AgentSpeak(L) can be converted into programs that can be run on the SIM_AGENT toolkit, resulting in the prototype interpreter SIM_Speak. SIM_Speak is tested on an application with a multi-agent traffic simulation.

Agent Communication

Since the issue of agent communication turned out to be a popular topic we have devoted a special part to it, although some of the papers might also have been put in one of the other parts.

Aiello et al. propose an agent architecture in which communication between two agents is facilitated by an overhearing agent that may intervene giving appropriate information. In particular, the authors give a formal language for the interaction between agents and overhearers where overhearing is based on ontological reasoning, that is, the overhearer selects pieces of communication according to his own ontologically organised knowledge and goals.

Next, *Amgoud and Parsons* present a general framework of dialogue where several agents can resolve differences of opinion and conflicts of interests, some of which result from differences in preferences.

Bonzon's paper addresses the problem of defining executable runs for classes of communicating agents. In particular, it focuses on agent classes of which the communication primitives are based on deduction.

Huber et al. consider a formal semantics for the interaction with middle agents that provide proxy services. To this end they define and analyze two new communicative acts, viz. PROXY and PROXY-WEAK, which result in the middle agents having significantly different levels of commitments relative to the final agents.

Yolum and Singh develop an approach in which they model communication protocols via so-called commitment machines, which supply a content to protocol states and actions in terms of the social commitments of the participants, and can be compiled into a finite state machine for efficient execution.

Collaborative Planning and Resource Allocation

This part is devoted to the problems of collaborative planning and resource allocation and has a highly algorithmic flavor.

The paper of *Hunsberger* presents algorithms that an agent may use to generate bids on sets of tasks in a proposed group activity in a combinatorial auction as part of a group decision-making mechanism. These algorithms generate temporal constraints for bids in such auctions that allow an agent making a bid to protect its private schedule of pre-existing commitments in case the bid is ultimately awarded.

Modi et al. propose a formalization of distributed resource allocation that represents both dynamic and distributed aspects of the problem at hand, explaining different sources of its difficulties. To this end the notion of Dynamic Distributed Constraint Satisfaction Problem is defined and a generalized mapping from distributed resource allocation to this problem is given.

Nuchia and Sen apply an algorithm that produces optimal envy-free divisions of continuously divisible goods between agents to improving the optimality of any envy-free division between an arbitrary number of agents. The proposed two-stage protocol first identifies all pair-wise envy-free exchanges and then utilizes a graph-based generate-and-test matching algorithm that selects the optimal set of such exchanges that will still produce envy-free allocations.

Trust and Safety

In this (small) part we have collected two papers, one dealing with evaluating the agent's trustworthiness, the other with the issue of safety in the context of a system with autonomous agents.

The paper by *Liu and Williams* investigates a procedure for evaluating an agent's trustworthiness as an information source. This process is split into competency analysis and sincerity analysis. An information pedigree is employed as a means to maintain the history of communicated information.

Pynadath and Tambe address the issue of adjustable autonomy of agents, also in team settings where agents must simultaneously satisfy safety team and individual responsibilities. Safety constraints are introduced that forbid or require certain states for actions in the context of Markov Decision Processes, and an algorithm is presented to propagate such constraints, which is proven correct and tested in a real-world domain.

Formal Theories of Negotiation

The topic of formal theories of negotiation between agents has been a special session at ATAL 2001. Since this part is introduced by its organiser *Frank Dignum* separately, we shall be very brief here:

Belmonte et al. present an analysis, based upon game theory, for a class of task-oriented problems arising from some internet transactions.

Felix Brandt and Gerhard Weiss present a strategy for bidders in repeated Vickrey auctions who are intending to inflict losses to fellow agents in order to be more successful, relatively to the group of bidders.

Esteva et al. present a specification language for institutions in electronic trading, covering norms, performative structure, scenes, roles, etc., together with a sketch of a formal semantics based on the Ambient Calculus.

Faratin et al. give a simple model of distributed multi-agent multi-issued contract negotiation for open systems where interactions are competitive and information is private and not shared.

Fatima et al. analyze the process of automated negotiation between two competitive agents that have firm deadlines and incomplete information about their opponent.

Littman and Stone consider agents that negotiate by issuing and responding to threats in the context of repeated, bimatrix games.

Sadri et al. present a formal, logic-based approach, based on agent dialogues, to one-to-one agent negotiation in the context of goal achievement in systems of agents with limited resource availability.

Agents for Hand-Held, Mobile, or Embedded Devices

Also the topic of designing and realizing agents for hand-held, mobile, or embedded devices has been a special session at ATAL 2001, organised by Tim Finin. We here briefly mention the three papers in this part.

Albuquerque et al. describe *KSACI*, a tool that provides communication infrastructure among agents running in handheld devices.

Bergenti and Poggi present the LEAP project aimed at realising a FIPA platform that can be deployed on any Java-enabled device with sufficient resources and with a wired or wireless connection.

Laukkanen et al. discuss the problem area of having a FIPA-OS agent platform running on small-footprint devices.

Knowledge Level Software Engineering

Fausto Giunchiglia[1], Anna Perini[2], and Fabrizio Sannicolò[1]

[1] Department of Information and Communication Technology
University of Trento
via Sommarive, 14
I-38050 Trento-Povo, Italy
fausto@dit.unitn.it
sannico@science.unitn.it
[2] ITC-Irst
Via Sommarive, 18
I-38050 Trento-Povo, Italy
perini@irst.itc.it

Abstract. We contend that, at least in the first stages of definition of the early and late requirements, the software development process should be articulated using *knowledge level* concepts. These concepts include *actors*, who can be (social, organizational, human or software) agents, positions or roles, *goals*, and *social dependencies* for defining the obligations of actors to other actors. The goal of this paper is to instantiate this claim by describing how *Tropos*, an agent-oriented software engineering methodology based on knowledge level concepts, can be used in the development of a substantial case study consisting of the meeting scheduler problem.

1 Introduction

In the last few years, many factors, noticeably the higher level of connectivity provided by the network technology and the ever increasing need of more and more sophisticated functionalities, have caused an exponential growth in the complexity of software systems. Examples of application areas where this is the case are e-commerce, e-business and e-services, enterprise resource planning and mobile computing. Software must now be based on open architectures that continuously change and evolve to accommodate new components and meet new requirements. Software must be robust and autonomous, capable of serving users with little or no computer expertise with a minimum of overhead and interference. Software must also operate on different platforms, without recompilations, and with minimal assumptions about its operating environment and its users.

The increased complexity calls for the development of new techniques and tools for designing, developing and managing software systems. We believe that the best way to deal with these problems is to analyze not only the *"what"* and the *"how"* of a software system, but also *"why"* we are using it. This is best done by starting the software development process with the early requirements, where one analyzes the domain within which the system will operate, by studying how the use of the system will modify the environment, and by progressively refining this analysis down to the actual implementation of the single modules. It is our further belief that this kind of analysis can be fruitfully done only by using *knowledge level* notions. In this context, we are using the term "knowledge level" in a technical sense, more precisely in the sense defined by Newell in his Turing Award Lecture [16]. Examples of knowledge level concepts are *actors*, who can be (social, organizational, human or software) agents, positions or roles, *goals*, and *social dependencies* for defining the obligations of actors to other actors. The use of knowledge level notions is necessary in order to analyze how the environment (consisting mainly of human actors) works. Using these notions also in the description of the software modules allows for a uniform and incremental refinement of the notions introduced in the early requirements [3]. From this perspective, agent oriented programming [14,1,23] has the advantage of allowing for the use of the same (knowledge level) concepts down to the actual implementation.

In previous papers [20,19,10,11] we have introduced *Tropos* [1], an agent-oriented software development methodology [8,22] based on the two key ideas hinted above, namely: *(i)* the use of knowledge level concepts, such as agent, goal, plan and other through all phases of software development, and *(ii)* a pivotal role assigned to requirements analysis when the environment and the system-to-be is analyzed. Tropos covers five software development phases: *early requirements analysis*, *late requirements analysis*, *architectural design*, *detailed design*, and *implementation.* A core workflow along the whole development process is the *conceptual modeling* activity, performed with the help of a visual modeling language which provides an ontology that includes knowledge level concepts. The syntax of the language is defined through a metamodel specified with a set of UML diagrams [20]. The language provides also a graphical notation for concepts, derived from the *i** framework [24] and a set of diagrams for viewing the models properties: *actor diagrams* for describing the network of social dependency relationships among actors, as well as *goal diagrams*, for illustrating goal and plan analysis from the point of view of a specific actor. A subset of the AUML diagrams proposed in [17,18] are adopted to illustrate detailed design specifications.

The goal of this paper is to instantiate our claim in favour of knowledge level software engineering by describing how Tropos can be used in the development of a substantial case study consisting of the *Meeting Scheduler* problem, as described in [21]. This problem is about the specification of software tools that can support the organization of a meeting, where the preferences of the important attendees are taken into account, as well as the constraints deriving from the other participants' agendas. (See [9,24] for alternative solutions to this problem.) The paper is structured as follows. The early requirements analysis is described in Section 2, the late requirements analysis in Section 3, and the architectural

[1] From the Greek "tropé", which means "easily changeable", also "easily adaptable". See also [6,5] for some early work on Tropos.

design in Section 4. Due to the lack of space, the last two phases, the detailed design and the implementation phases, are quickly summarized in Section 5. Each section starts with a short description of the activities of the phase described and then shows how these activities are instantiated in the solution of the Meeting Scheduler problem. The conclusions are presented in Section 6. A more detailed description of the last two phases can be found in [19].

2 Early Requirements

Early Requirements is concerned with the understanding of a problem by studying an existing organizational setting. During this phase, the requirements engineer models the stakeholders as actors and their intentions as goals. Each goal is analyzed from the point of view of its actor resulting in a set of dependencies between pairs of actors.

The stakeholders of the *Meeting Scheduler* domain, according to the problem statement given in [21], are the following:

- the *Meeting Initiator (MI)*, who wants to organize a meeting (such as a faculty meeting, a project meeting or a program committee meeting) in an effective way. In particular, he/she wants to make sure that the important participants will attend the meeting.
- the *Potential Participant (PP)*, that is, the generic participant, who would like to attend the meeting, possibly according to his/her preferences, or at least avoiding conflicts with other meetings.
- the *Important Participant (IP)*, who is a critical person for the objectives of the meeting. The meeting initiator will take care of checking preferences, both concerning the meeting location and the date, in order to assure his/her presence at the meeting.
- the *Active Participant (AP)*, is a meeting attendee who is required to give a presentation, so he/her possibly needs a overhead-projector or a workstation with or without network connection, and so on.

All these stakeholders can be modeled as roles that can be played by the same person. In particular, the meeting initiator can also play the role of the meeting participant. The complexity of the *Meeting Scheduler* problem is due to the fact that usually several meetings, sharing a set of participants, are going to be organized in parallel.

The initial early requirements model includes the stakeholders (roles) as actors and their intentions as goals. This model is illustrated with the actor diagram depicted in Figure 1, where actors are denoted as circles, goals as ovals and soft goals as cloudy shapes. Soft goals differ from goals because they don't have a formal definition, and are amenable to a different (more qualitative) kind of analysis. Soft goals are most often used to model the system non functional requirements (for instance software qualities, see [7] for a detailed description of soft goals). In particular the actor `MI`, the meeting initiator, has the goal `organize`

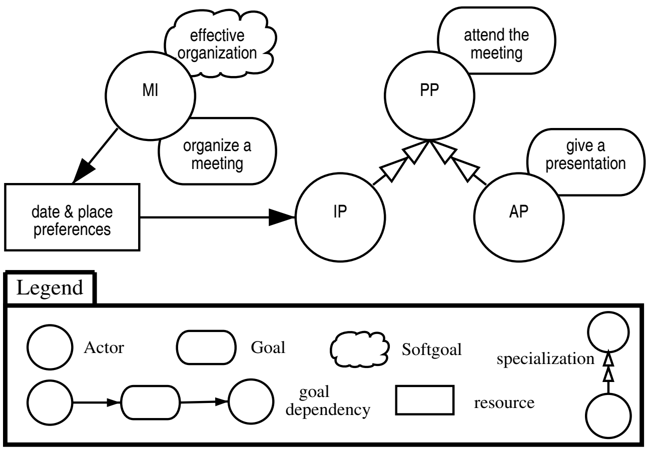


Fig. 1. *Actor diagram* where the social actors of the environment and their goals are modeled.

`a meeting`, possibly in an effective way. This latter objective is modeled as a soft goal, depicted as a cloudy shape labeled `effective organization`. The actor `PP`, models the potential attendee who has the goal `attend the meeting`. The actors `AP` and `IP`, that model the active participant and the important participant respectively, are modeled as a special type of potential participants. [2] That is, both have the goal `attend the meeting`, moreover, the actor `AP` has the goal `give a presentation`. Figure 1 shows also a resource dependency between the actor `MI` (the depender) and the actor `IP` (the dependee), concerning the information on `date & place preferences` of the important participant (the dependum). This dependency makes `MI` vulnerable, in the sense that if `IP` fails to deliver the dependum, then `MI` would be adversely affected in its ability to achieve the objective of arranging the meeting in such a way that the important participant attends it.

The early requirements analysis goes on by extending the actor diagram. This is done by incrementally adding more specific actor dependencies which come out from a goal and plan analysis conducted from the point of view of each actor. Figure 2 depicts a fragment of a goal diagram, obtained by exploding part of the diagram in Figure 1. Here, the perspective of `MI` is modeled. The diagram appears as a balloon within which `MI`'s goals are analyzed and his/her dependencies with the other actors are established. The goal `organize a meeting` is AND-decomposed into the subgoals `define meeting objectives`, `plan the meeting` and `identify participants`. Means to the achievement of the goal `plan the meeting` are the following two goals: `collect requirements`

[2] At the moment, it is still an open problem whether this kind of construct will appear in the final version of the methodology. The question is whether this construct is really a knowledge level construct or, rather, a software level construct that we are "forcing" in the early requirements analysis.

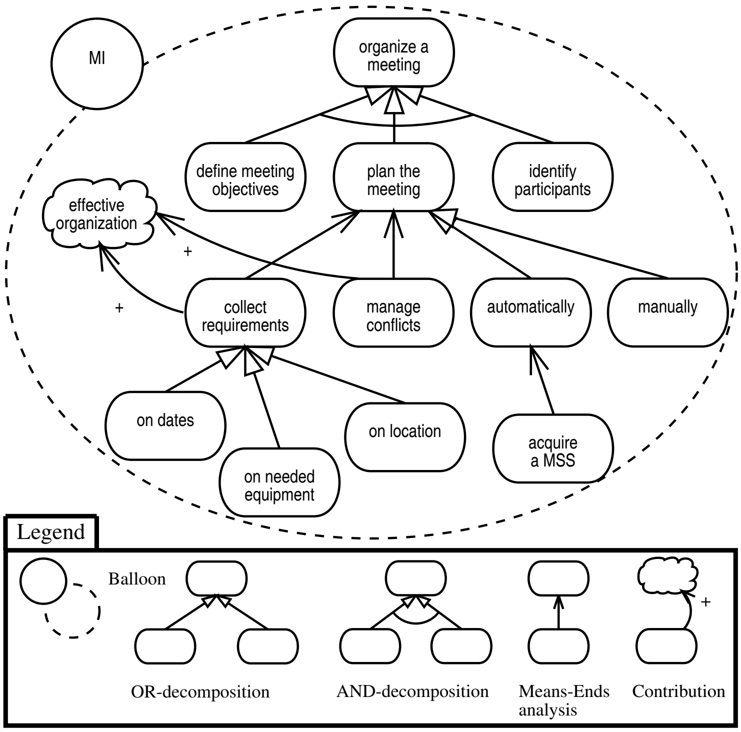


Fig. 2. Goal diagram for the actor `Meeting Initiator`, focusing on its goal `organize a meeting`.

and `manage conflicts`. The goal `collect requirements` refers to the need of asking the participants for the following information based on their personal agenda: dates on which they cannot attend the meeting (the "exclusion set"), and dates on which they prefer the meeting to take place (the "preference set"). Moreover, the active participants should specify the need of specific equipment for their presentation, while important participants could indicate their preference on the meeting location. So, the goal `collect requirements` has been further OR-decomposed into the goals `on dates`, `on needed equipment` and `on location`. According to [21], conflicts can be solved in several ways, such as extending the initial date range provided by the meeting initiator, asking participants to remove some dates from their exclusion set, asking important participant to add some dates to their preference set or withdrawing some participants from the meeting.

The goal `plan the meeting` could be achieved with the help of a software tool (see subgoal `automatically`) or manually (see subgoal `manually`). The goal `acquire a MSS`, that is the goal of acquiring a Meeting Scheduler System (MSS), is a means for achieving the goal of using a software tool. Both the goals `manage`

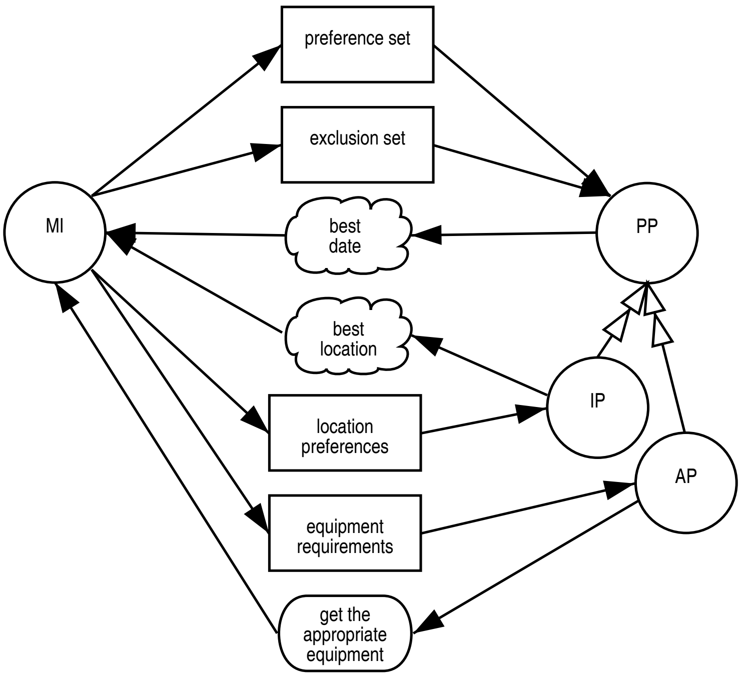


Fig. 3. Actor diagram showing the main dependencies between the social actors.

`conflicts` and `collect requirements` contribute positively to the fulfillment of the soft goal `effective organization`.

Figure 3 depicts the actor diagram resulting from the completion of the goal analysis of each social actor. `MI` depends on `PP`, the potential participant, for the information `preference set` and `exclusion set`, modeled as resources. Vice-versa, `PP` depends on `MI` for satisfying the soft goal `best date`. Moreover, `MI` depends on `IP` for getting preferences on the meeting location (see the resource dependency `location preferences`), and, vice-versa, `IP` depends on `MI` for the fulfillment of the soft goal `best location`. Finally, `MI` depends on `AP` for knowing the requirements on the specific equipment needed for the presentation, and `AP` depends on `MI` for the goal `get the appropriate equipment`.

3 Late Requirements

The Late Requirements phase analyses the system-to-be which is introduced as another actor into the model. The system actor is related to the social actors in terms of actor dependencies; its goals are analyzed and will eventually lead to revise and add new dependencies with a subset of the social actors (the users). When the scenario is sufficiently detailed, this provides a "use-cases" view.

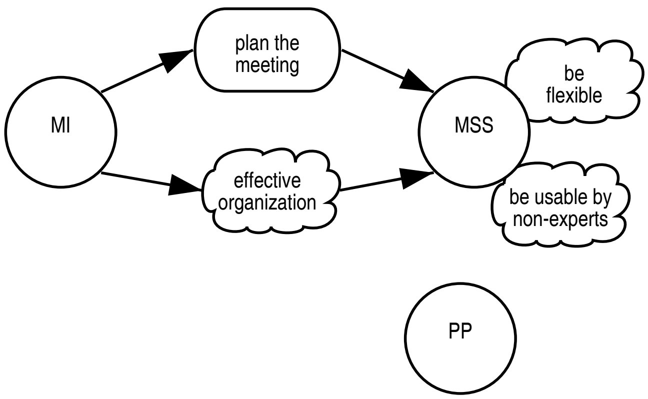


Fig. 4. Actor diagram where the system-to-be actor has been introduced.

The system-to-be, that is the *Meeting Scheduler System*, is represented by the actor `MSS`. Figure 4 illustrates the late requirements actor diagram. `MI` depends on the actor `MSS` for the goal `plan the meeting`, one of the `MI`'s subgoals discovered during the goal analysis depicted in Figure 2. The soft goal `effective organization` has been also delegated by `MI` to `MSS`. The dependencies between the social actors illustrated in Figure 2 need to be revised accordingly. The actor `MSS` has also two soft goals: `be flexible` and `be usable by non-experts`, which take into account some of the non-functional requirements described in [21].

The balloon in Figure 5 shows how the `MI`'s dependums can be further analyzed from the point of view of the *Meeting Scheduler System*. The goal `plan the meeting` is decomposed (AND-decomposition) into the subgoals `identify location`, `identify possible dates`, `communicate date & location` to the `IP`. The goal `plan the meeting` is a means for obtaining the goal `replan dynamically`, which is introduced as a goal of the actor `MSS`, according to the requirements of the *Meeting Scheduler System*. The goal `replan dynamically` contributes to the soft goal `be flexible`. The analysis proceeds by further decomposing goals into sub-goals and by identifying plans that can be considered as means for achieving these latter goals. So, for instance the goal `identify location` is AND-decomposed into the sub-goals `get equipment requirements`, `know location preferences`, `check location availability`, `solve possible conflicts`. Analogously, the goal `identify possible dates` is AND-decomposed into the sub-goals `manage conflicts`, `define exclusion sets` and `define preference sets`. The plan `get conflict management policies` from the meeting initiator has been introduced as a means for achieving the leaf goal. Analogously, the plans `get new exclusion set`, `get new preference set`, `get new location preferences`, are all means for achieving the goal `replan dynamically`.

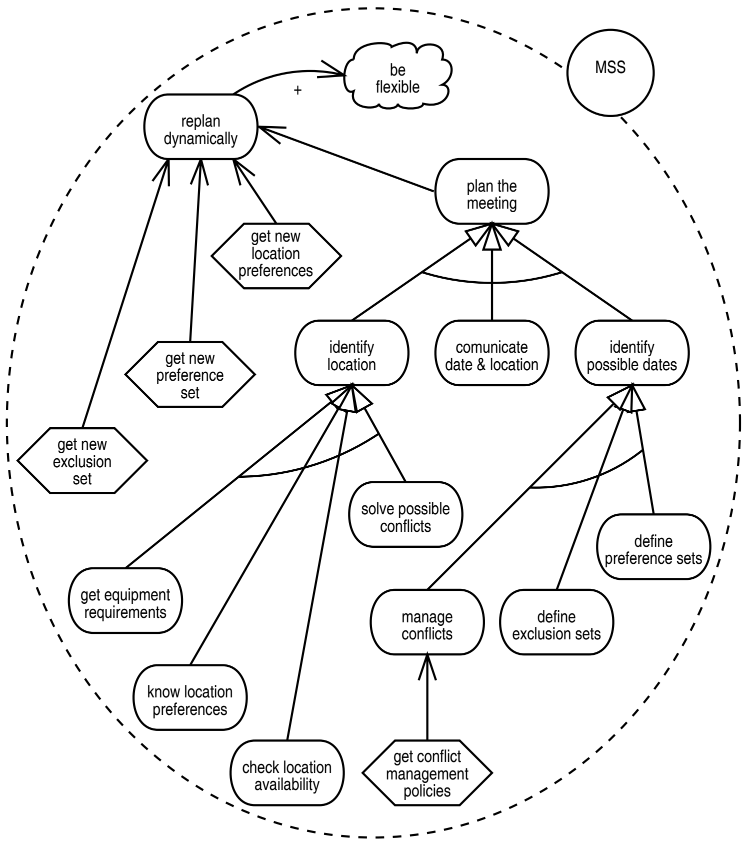


Fig. 5. Goal diagram of the system-to-be actor.

A set of dependencies between the system-to-be actor and the social actors can be derived from this analysis, as shown in the actor diagram depicted in Figure 6. So, the actor `MI` delegates to the system actor `MSS` the goal `plan the meeting`, while `MSS` depends on the actor `MI` for having a set of `conflict resolution policies` and general information about the meeting, such as the meeting objectives and information necessary to identify potential, important and active participants. Moreover, the plans identified in the goal analysis of the `MSS` actor motivates a set of resource dependencies among the system actors and the actors modeling the different roles of the meeting participants, that is `date preferences`, `location preferences`, `exclusion set`, `equipment requirement`. Vice-versa, the following resource dependencies link these actors and the `MSS` actor: `new date preference`, `new location preference`, `new exclusion set` and `new equipment requirement`. Finally, `PP` depends on `MSS` for being promptly advised about the `final date & location` of the meeting. The resulting actor diagram sketches a "use-case"

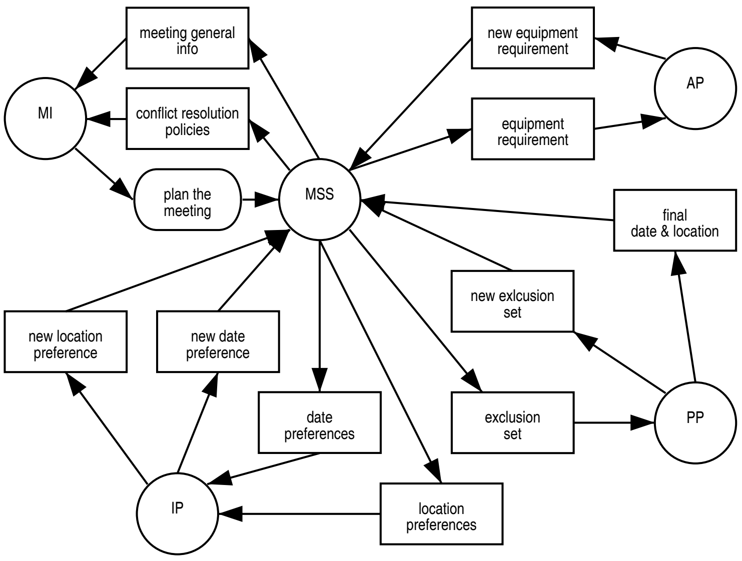


Fig. 6. Actor diagram resulting from the late requirements analysis.

diagram [13]. Notice how here use-cases are obtained as a refinement of the Early and Late requirements Analysis. Notice also how this is not the case for UML [2] where use-case diagrams are an add-on which hardly integrates with the other diagrams.

4 Architectural Design

Architectural design defines the system's global architecture in terms of subsystems, interconnected through data and control flows. Subsystems are represented as actors and data/control interconnections are represented as (system) actor dependencies. This phase consists of three steps:

1. *refining the system actor diagram,*
2. *identifying capabilities and*
3. *assigning them to agents.*

Step 1. The actor diagram is refined adding new sub-systems, according to the following sub-steps:

a inclusion of new actors due to the delegation of sub-goals, upon goal analysis of the system's goals;
b inclusion of new actors according to the choice of a specific architectural style agent (design patterns [15]);
c inclusion of new actors contributing positively to the fulfillment of some non-functional requirements

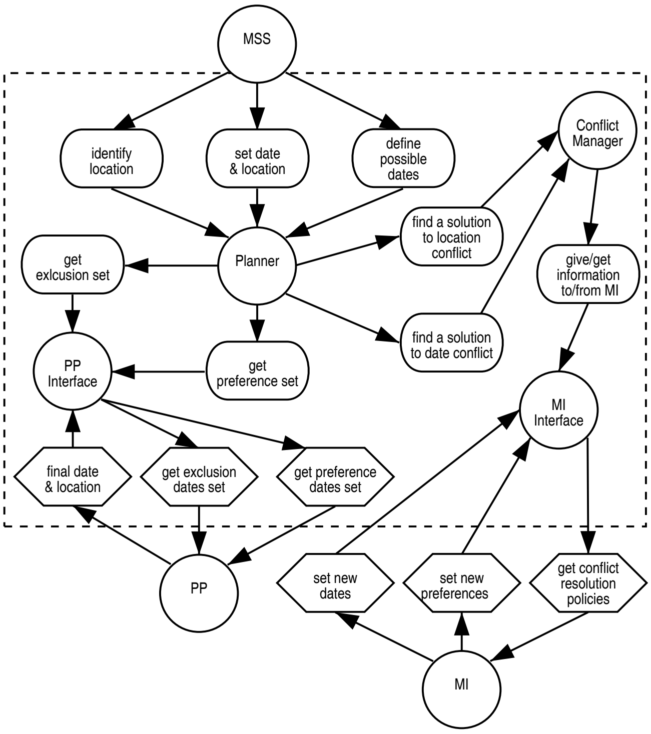


Fig. 7. A Portion of the architectural design model for the system-to-be. Extended actor diagram.

Here, we focus on an example of step 1.a. A portion of the system architecture model is illustrated in the actor diagram depicted in Figure 7. Four sub-system actors have been introduced. The first is the `Planner` to which `MSS` delegates the following goals: `identify location`, `define possible dates`, `set date & location`. The `Planner` rests on the actor `PP Interface` for the achievement of the goals `get exclusion set`, `get preference set`, and on the actor `Conflict Manager` for the achievement of the goals `find a solution to location conflict` and `find a solution to date conflict`. Moreover, the actor `Conflict Manager` rests on the actor `MI Interface` to satisfy its goal of giving and getting information to/from the meeting initiator. Figure 7 includes also some dependencies with the social actors interacting with the system. In particular, the actor `PP Interface` depends on the social actor `PP`, to have the plans of getting exclusion and preference dates set, while `PP` depends on the actor `PP Interface` to receive information on the meeting (such as final date and location). Analogously, the actor `MI Interface` depends on the social actor `MI` to execute the plans of getting conflict resolution policies, and, vice-versa, `MI` depends on the actor `MI Interface` to be be able to set new preferences and new dates for the meeting to be organized.

Table 1. Actors capabilities.

Actor Name	N	Capability
Planner	1	identify location
	2	define possible dates
	3	set date & location
	4	get exclusion set
	5	get preference set
	6	find a location conflict solution
	7	find a date conflict solution
PP Interface	8	provide exclusion set
	9	provide preference set
	10	interact with PP to get exclusion set
	11	interact with PP to get preference set
Conflict Manager	12	resolve a conflict
	13	give/get information to/from MI
	14	provide solutions to location conflicts
	15	provide solutions to date conflicts
MI Interface	16	get & provide info to Conflict Manager
	17	interact with MI to get resolution policies
	18	interact with MI to get new dates
	19	interact with MI to get new preferences

The system architecture model can be further enriched with other system actors according to design patterns [12] that provide solutions to heterogeneous agents communication and to non-functional requirements, as also described in [10].

Step 2. The actor capabilities are identified from the analysis of the dependencies going-in and -out from the actor and from the goals and plans that the actor will carry on in order to fulfill functional and non-functional requirements.

Focusing on the system actor `Planner` depicted in Figure 7, and in particular on its ongoing and outgoing dependencies we can identify the following capabilities: `identify location`, `define possible dates`, `set date & location`, `get exclusion set`, `get preference set`, `find a location conflict solution`, `find a date conflict solution`. Table 1 lists the capabilities associated to the actors depicted in the diagram of Figure 7.

Step 3. Agent types are defined. One or more different capabilities are assigned to each agent.

In general, the agent assignment is not unique and depends on the designer experience. Table 2 shows an example of agent assignment on a subset of the

Table 2. Agent types and their capabilities. An example.

Agent	Capabilities
Planner	1, 2, 3, 4, 5, 7
User Interface	6, 8, 9, 10, 11, 16, 17, 18, 19
Conflict Manager	12, 13, 14, 15

system-to-be actors. This table shows that there is more than one mapping from actors and agent types. Other associations could be easily find out.

5 The Last Two Phases: Detailed Design and Implementation

Detailed design aims at specifying the agent microlevel, defining capabilities, and plans using the AUML activity diagram, and communication and coordination protocols using the AUML sequence diagrams. A mapping between the Tropos concepts and the constructs of the implementation language and platform is provided.

Figure 8 depicts, as an example, the capability diagram of the `manage location conflict`, one of the capabilities of the agent `Conflict Manager`. The capability is triggered by the external event corresponding to the request of the agent `Planner` to the `Conflict Manager` of managing a location conflict, given a list of preferred locations and a list of available time periods for each location. The agent `Conflict Manager` chooses one of the possible plan corresponding to different conflict management policies. For instance, `policy 1` corresponds to the plan which, for a given date, checks the availability of the meeting location, according to the stated preference order. If there is no solution the `Conflict Manager` tries a policy that has not yet been used or asks the user for a solution, via the `User Interface actor`. The plan `policy 2` checks, for a given location, the dates at which the location is not allocated, taking into account the preferred date first. If there is no solution the `Conflict Manager` tries a policy that has not yet been used or asks the user for a solution, via the `User Interface` agent. The third policy corresponds to the plan where the user is asked to: (*i*) sort the preferred dates, (*ii*) sort the preferred locations, (*iii*) choose which policy, among `policy 1` and `policy 2` should be tried first[3]. The solution is noticed back to the agent `Planner`.

[3] *Plan diagrams*, namely AUML activity diagrams, for the given plans can be straightforwardly derived. They are not included here due to lack of space. For the same reason we can not show here the specification of agent interactions via AUML sequence diagram, see [19] for an example.

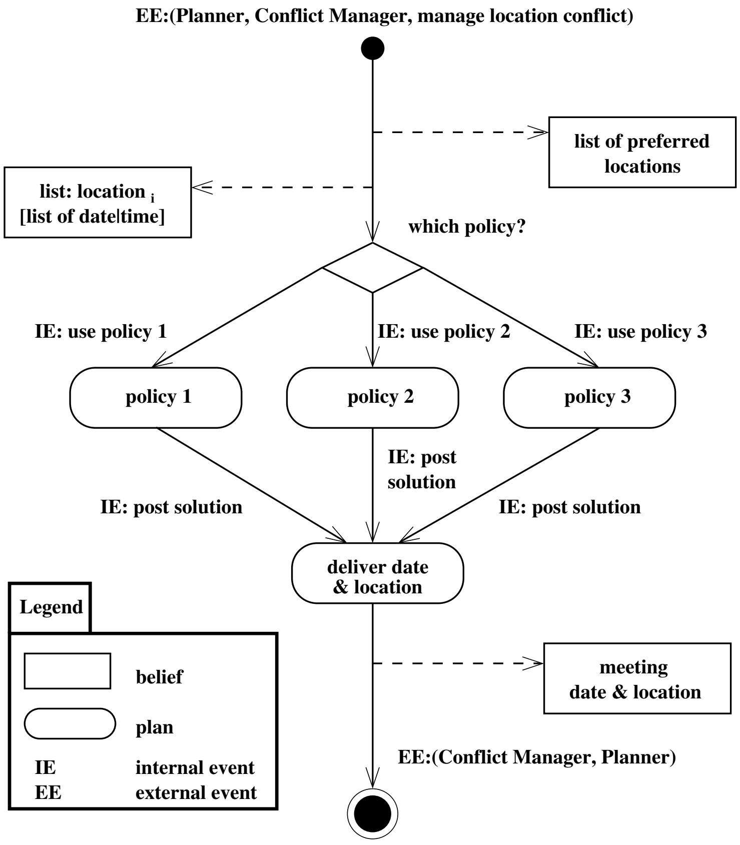


Fig. 8. *Capability diagram* of `manage location conflict`, one of the capabilities of the actor `Conflict Manager`.

Finally, the implementation activity follows step by step, in a natural way, the detailed design specification which is transformed into a skeleton for the implementation.

This is done through a mapping from the Tropos concepts to the constructs of the programming language provided by the implementation platform that has been chosen before starting the detailed design phase. An example, based on the use of the BDI agent programming platform JACK [4], can be found in [19].

6 Conclusions

In this paper we have applied the Tropos methodology to the case study of the Meeting Scheduler problem. Tropos is a new software development methodology, mainly designed, but not only, for agent based software systems, which allows us to model software systems at the knowledge level. The main goal of this paper has been to show, via an example, how, by working at the knowledge level, we can cope with the increased complexity that many new applications require.

Acknowledgments

We thank all the Tropos Project people working in Trento and in Toronto.

References

1. FIPA. Foundation for Intelligent Physical Agents. http://www.fipa.org.
2. G. Booch, J. Rambaugh, and J. Jacobson. *The Unified Modeling Language User Guide*. The Addison-Wesley Object Technology Series. Addison-Wesley, 1999.
3. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Modeling early requirements in tropos: a transformation based approach. In Wooldridge et al. [22].
4. P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents - Components for Intelligent Agents in Java. Technical Report TR9901, AOS, January 1999. http://www.jackagents.com/pdf/tr9901.pdf.
5. J. Castro, M. Kolp, and J. Mylopoulos. Developing agent-oriented information systems for the enterprise. In *Proceedings Third International Conference on Enterprise Information Systems*, Stafford UK, July 2000.
6. J. Castro, M. Kolp, and J. Mylopoulos. A requirements-driven development methodology. In *Proc. 13th Int. Conf. on Advanced Information Systems Engineering CAiSE 01*, Stafford UK, June 2001.
7. L. K. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.
8. P. Ciancarini and M. Wooldridge, editors. *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in AI*. Springer-Verlag, March 2001.
9. R. Darimont, A. van Lamsweerde, and P. Massonet. Goal-Directed elaboration of requirements for a meeting scheduler: problems and lesson learnt. In *Proceedings RE'95 - 2nd IEEE Symposium on Requirements Engineering*, pages 194–203, York, March 1995.
10. P. Giorgini, A. Perini, J. Mylopoulos, F. Giunchiglia, and P. Bresciani. Agent-oriented software development: A case study. In S. Sen J.P. Müller, E. Andre and C. Frassen, editors, *Proceedings of the Thirteenth International Conference on Software Engineering - Knowledge Engineering (SEKE01)*, Buenos Aires - ARGENTINA, June 13 - 15 2001.
11. F. Giunchiglia, J. Mylopoulos, and A. Perini. The Tropos Software Development Methodology: Processes, Models and Diagrams. Technical Report No. 0111-20, ITC - IRST, Nov 2001. Submitted to AAMAS '02.
12. Sandra Hayden, Chirstina Carrick, and Qiang Yang. Architectural design patterns for multiagent coordination. In *Proc. of the International Conference on Agent Systems '99*, Seattle, WA, May 1999.
13. Ivar Jacobson, Mangus Christerson, Patrik Jonsson, and Gunmar Övergaard. *Object-Oriented Software Engineering: a Use-Case Driven Approach*. Addison Wesley, Readings, MA, 1992.
14. N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2), 2000.

15. M. Kolp, P. Giorgini, and J. Mylopoulos. An goal-based organizational perspective on multi-agents architectures. In *Proc. of the 8th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, Seattle, WA, August 2001.
16. A. Newell. The Knowledge Level. *Artificial Intelligence*, 18:87–127, 1982.
17. J. Odell and C. Bock. Suggested UML extensions for agents. Technical report, OMG, December 1999. Submitted to the OMG's Analysis and Design Task Force in response to the Request for Information entitled "UML2.0 RFI".
18. J. Odell, H. Parunak, and B. Bauer. Extending UML for agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Proc. of the Agent-Oriented Information Systems workshop at the 17th National conference on Artificial Intelligence*, pages 3–17, Austin, TX, 2000.
19. A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini, and J. Mylopoulos. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In *Proc. of the 5th Int. Conference on Autonomous Agents*, Montreal CA, May 2001. ACM.
20. F. Sannicolo', A. Perini, and F. Giunchiglia. The Tropos modeling language. a User Guide. Technical report, ITC-irst, December 2001.
21. A. v. Lamsweerde, R. Darimont, and P. Massonet. The Meeting Scheduler System - Problem Statement. Technical report, Université Catholique de Louvain - Département d'Ingénierie Informatique, B-1348 Louvain-la-Neuve (Belgium), October 1992.
22. M. Wooldridge, P. Ciancarini, and organizers G. Weiss, editors. *Proc. of the 2nd Int. Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, Montreal, CA, May 2001.
23. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.
24. E. Yu. *Modelling Strategic Relationships for Process Reengineering.* PhD thesis, University of Toronto, Department of Computer Science, 1995.

Emotions and Personality in Agent Design and Modeling

Piotr J. Gmytrasiewicz[1] and Christine L. Lisetti[2]

[1] Computer Science Department
University of Illinois at Chicago
Chicago, IL 60607-7053
piotr@cs.uic.edu

[2] Department of Computer Science
University of Central Florida
Orlando, FL 32816-2362
lisetti@cs.ucf.edu

Abstract. Our research combines two diverse strands of work in AI and cognitive science. We start from the principled paradigm of rational agent design based on decision theory. We then use this paradigm to formally define the emotional states and personality of an artificial intelligent agent. We view the emotional states as the agent's decision-making modes, predisposing the agent to make its choices in a specific, yet rational, way. Change of the emotional state, say due to an external stimulus, invokes a transformation of the agent's decision-making behavior. We define personality as consisting of the agent's emotional states together with the specification of transitions taking place among the states. To enable an artificial agent to model the personalities and emotional states of agents and humans that it interacts with, we additionally provide a precise definition of a personality models of other agents. Our definition allows the personality models to be learned over the course of multiple interactions with the users and other agents.

1 Introduction

The objective of this research is to develop a fundamental understanding of the role and usefulness of the notions of emotions and personality in designing rational artificial agents. The main hypothesis of our work is that notions of personality and emotions are useful in designing competent artificial agents that are to operate within complex uncertain environments populated by other artificial and human agents. Our work draws on and combines an emerging technology of rational agent design from artificial intelligence on the one hand [7,9,28,34], with research on human emotions in cognitive science and psychology on the other hand [8,11,15,16,17,20,25,30,32,31].

Our work accepts the decision-theoretic paradigm of rationality, according to which a rational agent should behave so as to maximize the expected utility of its actions (see [3,9,28] and other references therein). The expected utilities of the alternative courses of action are computed based on their possible consequences, the desirability of these consequences to the agent,[1] and the probabilities with which these consequences are thought by the agent to obtain.[2] We aim to examine ways in which components of the

[1] Such agents are sometimes called self-interested.
[2] The probabilities are, therefore, subjective.

decision-theoretic model, i.e., the utility functions, the set of behavioral alternatives, and the probabilities of consequences, can be transformed in ways that has been recognized in cognitive science as interactions between emotional states and decision-making. Further, we want to make the personality and the emotional components useful to model other agents, including humans, during interactions.

As research in cognitive science shows, some of the most important functions of emotions is to manage the individual's cognitive resources, allow him to understand the other agents' internal states, and to effectively communicate his internal state to others. The importance of managing the cognitive resources is clear when one considers the cognitive limitations of animals, humans and machines on the one hand, and the computational demands imposed by complex dynamical environments on the other hand. Given these limitations, biological systems are equipped with simplified ways of arriving at the appropriate action to be executed. Lower-level mechanisms, such as instincts, are essentially condition-action rules. Higher-level mechanism is provided by emotions, which dictate not actions but action tendencies [17,30,32]. For example, if an external event threatens the safety of an individual, the emotional state of fear restricts the alternative behaviors to a small repertoire of possible appropriate actions such as: interrupt current activity, monitor environment, flee, or fight. In this context, emotional states are modifying the parameters of deliberative rationality to control its complexity under time pressure. The significance of our work lies in the formal definitions of these functions and their use for the benefit of the artificial interactive agents we are designing.

As we mentioned, the other aspect in which emotions and personality are important is the ability to understand and appropriately model the internal states of other agents. For example, during human-computer interactions the users' emotional state, such as anger, fear, boredom, panic, surprise, joy, or excitation, can be assessed using measurable factors (facial expression recognition, vocal intonation, prosody, galvanic skin response, heart rate and breathing patterns, haptic and tactile feedback, body posture [6,14,21,22,26]). The availability of these technologies clearly brings forth the need for a principled understanding of how to interpret these signals, and how to use them to predict the user's needs and actions. Without this understanding the machine may forever be hopelessly out-of-step with the emotional state of the human user. Our definitions of emotional states allow the system to model the user in such cases, and to robustly interpret the signals indicating the user's emotions. Further, our definition of personality introduces a principled dynamic model of the user's emotional states. Using it enables a prediction that, for example, a user already annoyed will not be calmed down by another system response that is not along the user's wishes. Even if the machine is interacting with an artificial agent, the ability to understand and model the agent's internal state can be of essence. For example, effective interaction with a bounded agent operating under time pressure requires the understanding that the agent probably does not have time and computational resources required to evaluate all of the available options in great detail.

Closely related to the above is the need for reliable and universally understood vocabulary using which the agents can communicate their internal states to each other. For example, it may be more expeditious to communicate that an agent is "panicked" than to explain at length that the time horizon of the alternative plans being considered had to be shortened due the computational limitations and time pressure of the situation at hand. Also, the lengthy explanation may be meaningful only to agents of similar designs. Thus, a formal understanding of what emotions are, in terms of already known theories of behavior, serves as semantics of communicative terms that describe the agents' internal states.

As we mentioned, our approach complements and builds on the existing approaches to designing rational and socially competent agents [2,3,5,9,10,12,13,18,27,28,33,34]. Such agents should be able to function efficiently under time and other environmental pressures, and be able to interact and communicate with other agents. This includes informing each other about details of the external environment and about the agents' own internal states, as well as the ability to model and predict the internal states of other agents. Apart from the area of multi-agent systems, our approach has applications in Human-Computer Interaction (HCI) that range from intelligent tutoring systems and distance learning support systems (with recognition of expressions signaling interest, boredom, confusion), to stress and lie detectors, to monitors of pilots' and drivers' state of alertness, to software product support systems (with recognition of users being dis/pleased with software products), to entertainment and computer games (enjoyment, confusion), and to ubiquitous computing and smart houses [19].

2 Decision-Theoretic Preliminaries

As we mentioned, the objective of our research is to develop a fundamental understanding of the role and usefulness of emotional states and personality in designing intelligent artificial systems. Our approach draws on and combines an emerging technology of rational agent design of Artificial Intelligence on the one hand [3,7,9,28,34], with research on human emotions in cognitive science and psychology on the other hand [8,15,16,17,20,25,30,32,31].

We use the decision-theoretic paradigm of rationality, according to which a rational agent should behave so as to maximize the expected utility of its actions (see [3,9,28] and references therein). The expected utilities of the alternative courses of action are computed based on their possible consequences, the desirability of these consequences to the agent, and the probabilities with which these consequences are thought by the agent to obtain.

Thus, a rational agent formulates its decision-making situation in terms of a finite set, A, of the alternative courses of action, or behaviors, it can execute, which we will call the agent's *action space*. An alternative behavior, say a_i, is a plan consisting of consecutive actions extending into the future time t_{a_i}, which we will call the time horizon of this particular plan. Alternative courses of action in set A can stand for abstract actions as well as for detailed elaborations; increasing the level of abstraction facilitates keeping the size of A down to manageable proportions. We demand that the actions be distinct and that the set A be exhaustive, i.e., that all of the possible behaviors be accounted for. Sometimes an "all-else" behavioral alternative is used for compactness, and represents all other possible behaviors except the ones explicitly enumerated.

At any point, an agent finds itself in some state of the world, but due to the fact that the environment may not be fully observable the agent may be uncertain about the state. The fact that the actual state may be unknown to the agent can be formalized by specifying the set of all possible states of the world, S, together with a family of probability distributions, $\mathbf{P}(S)$, over these states. One of these distributions, say $P_c(S)(\in \mathbf{P})$, specifies which of these states are currently possible and how likely they are. Thus $P_c(S)$ fully describes the information the agent has about the present state of the world.

The agent can ponder the consequences of its alternative actions. Due to possible nondeterminism each action, $a_i \in A$, may lead to many resulting possible states. The likelihoods of the resulting states can be specified by another probability distribution, $P_i(S)(\in \mathbf{P})$, also over S. The process of determining the probabilities of different results, i.e., the distribution P_i has been called a probabilistic temporal projection. The projection is a function $Proj : \mathbf{P}(S) \times A \rightarrow \mathbf{P}(S)$, so that the result of projecting the results of action a_i, given the current information about the state $P_c(S)$, results in the projected information about the resulting state, $P_i(S)$: $Proj(P_c(S), a_i) = P_i(S)$. The above formulation does not preclude state changes due to actions of other agents or erogenous events; here these effects are implicit and folded into the projection function [3].

The desirabilities of the states of the world to the agent are encoded using a utility function $U : S \rightarrow \Re$, which maps states of the world to real numbers. Intuitively, the higher the utility value of a state the more desirable this state is to the agent. The agent decision problem involves choosing which of the alternative actions in the set A it should execute. One of the central theorems of decision theory states that if the agent's utility function is properly formed, and the agent expresses its uncertain beliefs using probabilities, then the agent should execute an action, a^*, that maximizes the expected utility of the result.

$$a^* = ArgMax_{a_i \in A} \sum_{s^j \in S} p_i^j U(s^j), \tag{1}$$

where the p_i^j is the probability the projected distribution $P_i(S)$ assigns to a state $s^j \in S$.[3] Frequently, it is convenient to represent the utility function, U, as depending on a small number of attributes of the states of the world, as opposed to depending on the states themselves. This is intuitive; humans may prefer, say, all of the states in which they have more money, are more famous, and are healthier. The attributes of wealth, fame, and health are then convenient factors in terms of which the utility function can be expressed. Multi-attribute utility theory postulates that, in some simple cases, the utility of a state be a weighted sum of the utilities, $U(X_l(s))$ of individual attributes:

$$U(s) = \sum_{X_l \in Attributes} W_{X_l} U(X_l(s)), \tag{2}$$

where the W_{X_l} is the weight or, intuitively, the importance, of the attribute X_l. Having the weights of the attributes explicitly represented is convenient since it enables the trade offs among the attributes the agent may have to make. For example, the agent may have to give up some of its wealth to improve its health, and so on.

[3] Eq. 1 is exact only for simple nonsequential decision making with one step look-ahead. For sequantial decisions one needs to replace the $U(s^j)$ with a value function, $V(s^j)$, that includes the benefits of subsequent actions that can be executed in state s^j. Thus, V is: $V(s) = U(s) + max_{a_i \in A} \sum_{s^k \in S} p_j^k V(s^k)$; see [3,28] for details.

The elements defined above are sufficient to formally define a decision-making situation of an agent:

Definition 1: A decision-making situation of an agent is a quadruple: $D = \langle P_c(S), A, Proj, U \rangle$, where S, $P_c(S)$, A, $Proj$ and U are as defined above.

The above quadruple fully specifies the agent's knowledge about the environment, the agent's assessment as to its possible courses of action, the possible results of the actions, and desirability of these results. Our definition here is a version of partially observable Markov decision process (see [3,28] and references therein), but it makes explicit the time horizons of the alternative action sequences the agent is choosing among.

Given its decision-making situation, an agent can compute its best action, a^*, as specified in Equation 1. It is clear that this computation can be fairly complex. In a multi-agent environment, for example, all of the information the agent has about the physical environment and about the other agents could be relevant and impact the expected utilities of alternative courses of action. Sometimes the agent may have information about the other agents' state of knowledge, which is also potentially relevant. Given these complexities it is clear that a mechanism for managing the agent's computational resources is needed. In our work, we exploit emotional states and personality, as defined below, as a tool providing for such ability.

3 Emotional States and Personality

As we mentioned, we view emotional states as different modes of decision-making, defined above. The set of emotional states and transitions between them comprise the agent's personality, which we define further below.

A simple taxonomy [25] of emotional states in Figure 1 is aimed at differentiating among emotional states by using values of well-defined attributes. In its leaves, the taxonomy includes the basic, also called primitive [24], emotions of happiness, sadness, fear and anger. Each of these predisposes the agent to make decisions in a different way, and we associate each of them with a different decision-making situation:

Definition 2: An emotional state of an agent is associated with its decision-making situation $D = \langle P_c(S), A, Proj, U \rangle$, defined above.

The above definition *associates* an emotional state with a decision-making situation, but it does not *identify* the two to be the same. The reason is that emotions, apart from being decision-making modes, may involve physical and other changes (say breathing rate, muscle tension, dilation of the pupil or rate of power consumption.) However, this definition does allow us to describe different emotions in terms of the elements of the decision-making situation, and naturally illustrates the impact emotions have on an agent's behavior. For example, an emotional state characterized by a low utility U assigned to all states, which could be called sadness, will have a well-defined effect on the behavior the agent will arrive at. We detail several other ways of modifying the elements of the decision-making situation that can be associated with various emotions later. Now we go on to define personality.

We conceptualize an agent's personality as the set of emotional states the agent is capable of being in, the transformations that are possible between these states, and the events that trigger the transformations.

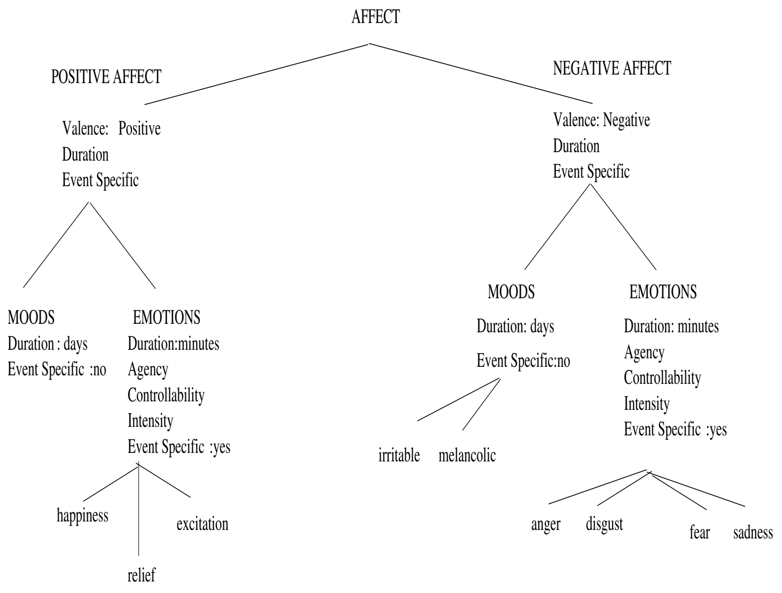


Fig. 1. An Example Taxonomy of Emotional States.

Definition 3: An agent's personality is a finite state machine $P = \langle \mathbf{D}, \mathbf{IN}, \Delta, N \rangle$, where

- **D** is a finite set of emotional states, defined above,
- **IN** is a set of environmental inputs,
- Δ is an emotional transformation function, $\Delta : \mathbf{D} \times \mathbf{IN}^* \rightarrow \mathbf{D}$,
- $N \in \mathbf{D}$ is an initial (or neutral) emotional state.

The above definition specifies that the agent's emotional state can change due to a, possibly empty, sequence of environmental inputs. This means that emotions are passive - they happen to the agent without the agent's control. As we mentioned, in biological systems they are built-in and are of essential survival value in emergency situations that are too important to rely on cognition [24]. In artificial systems, the personality construct we defined above is a method controlling deliberative decision-making imposed by the agent's designer. The major challenge is to establish what kinds of personalities are most beneficial, given the range of environments the agent is to operate in.

The definition of personality above formalizes this notion from the individual agent's point of view, i.e., it describes the personality of the agent itself. To equip the agent with a tool with which to model emotional states of other agents we define a related notion of a personality model.

Definition 4: A personality model of agent R is a probabilistic finite state machine $P_R = \langle \mathbf{D}, \mathbf{IN}, \Delta, N \rangle$, where

- **D** is a finite set of emotional states of agent R,
- **IN** is a set of environmental inputs,

– Δ is a probabilistic transformation function, $\Delta : \mathbf{D} \times \mathbf{IN}^* \times \mathbf{D} \rightarrow [0, 1]$,
– $N \in \mathbf{D}$ is an initial (or neutral) emotional state of agent R.

An agent that has a personality model of another agent, R, can use this model to probabilistically predict R's emotional state, given an initial state and an environmental input. The transformation function is probabilistic to allow for uncertainty as to the next emotional state of the modeled agent. It assigns a probability $\Delta(N, IN, D)$ to a possible transformation between emotional states N and D due to an environmental input IN. The main advantage of using this approach is that a personality model can be learned, given limited amount of observations of the other agent's behavior, using an unsupervised US-L learning algorithm (see, for example [4,23,29].)

In Figure 2 we present a very simple example of a personality model. In it, the transformation function happens to be Boolean, and the agent is capable in being in one of only three emotional states: COOPERATIVE, SLIGHTLY ANNOYED, and ANGRY. The transitions among the states are caused by environmental inputs which are divided into Cooperative and Uncooperative ones. Using this dynamic model one can predict that an agent that is in COOPERATIVE emotional state will become SLIGHTLY ANNOYED given Uncooperative input. Further, the emotional state of SLIGHTLY ANNOYED will evolve into ANGRY if another Uncooperative response follows.

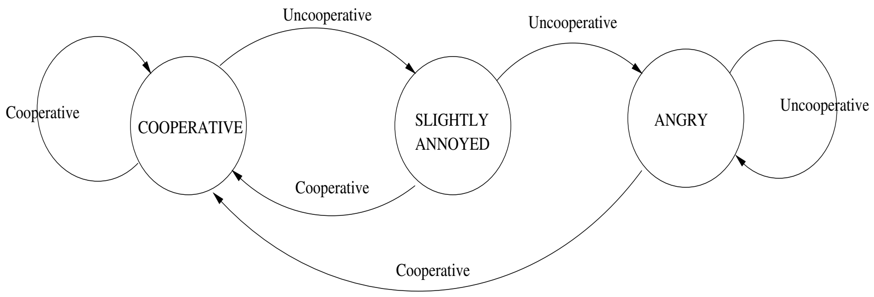


Fig. 2. Simple Personality Model of a Tit-for-Two-Tats Agent.

In Figure 2, if the COOPERATIVE state is the initial (neutral) state, then this personality model corresponds to the Tit-for-Two-Tats strategy widely investigated in the literature on repetitive prisoner's dilemma game [1,4,29]. Here, all three emotional states radically change the agent's decision-making model by limiting the alternative behaviors to cooperative (executed in the COOPERATIVE and SLIGHTLY ANNOYED states) to uncooperative (executed in the ANGRY state.)

We now present an initial approach we have taken to defining classes of emotional states in terms of the elements of the agent's decision-making situation. We find it intuitive to describe them as differing from, or transforming, the elements defining the "neutral" emotional state: $N = \langle P_c(S), A, Proj, U \rangle$.

3.1 Transformations of the Action Space A

Transformation of the action space A, for example by narrowing the set of alternative actions considered to encompass only a small subset of all of the actions, predisposes the agent to take action from this smaller set. This constitutes the *action tendency* that the emotion is invoking in the agent, as postulated, for example, by Fridja in [11]. In the extreme, narrowing the set A to a single action implements a behavioral condition-response rule, as in the emotional states COOPERATIVE, SLIGHTLY ANNOYED, and ANGRY of the Tit-for-Two-Tats agent above.

Formally, these are emotional transformations $\Delta(N, IN) = D$ such that $N = \langle P_c(S), A, Proj, U\rangle$, $D = \langle P_c(S), A', Proj, U\rangle$. The new emotional state, D, corresponds to an action tendency if $A' \subset A$. For example, an agent becoming angry may result in it considering only a subset of its behavioral alternatives – the ones of aggressive nature. A special case is when A' is a singleton set, as in the Tit-for-Two-Tats above.

Another intuitive special case of such transformation is one that results in the agent's deliberating in a more short-term fashion, such as it being rushed or panicked under time pressure. Formally we have: $\forall a_i^{'} \in A' : t_{a_i^{'}} \leq t_{a_i}$, which states that the time horizon of alternative plans considered has diminished. This is characteristic of human decision-makers; people frequently become more short-sighted when they are rushed or panicked, since they have no time to consider long-term effects of their alternative behaviors.

3.2 Transformations of the Utility Functions U

Intuition behind this transformation is that emotions both implement U, as well as modify it. Humans evaluate desirability of states by having positive or negative feelings about them. Positive or negative emotions or moods may alter these evaluations by, say, decreasing them, as in melancholic or depressed moods (when everything looks bleak), or increasing them, as in elated or happy moods. Other emotional states can change the weights of the factors contributing to the utility ratings (Equation 2). The agent's behavior resulting from these transformations may be one of inaction, when all of the behaviors available lead to seemingly worthless results, or recklessness, when some factor regarded as important previously in the utility function has its weighting reduced.

Formally, these are transformations $\Delta(N, IN) = D$ such that $N = \langle P_c(S), A, Proj, U\rangle$, and $D = \langle P_c(S), A, Proj, U'\rangle$. A special case of sadness is when the desirability of every state diminishes: $\forall s \in S : U'(s) \leq U(s)$.

3.3 Transformations of the Probabilities of States

The intuition behind this transformation is that changing these probabilities, for instance by simplifying them, can be helpful and save time under time pressure. The most radical simplification is one that makes the most likely state to be the only possible state. This corresponds to considering only the most likely result of action and neglecting all less likely states, which is often observed in human decision-makers.

Formally, these are transformations $\Delta(N, IN) = D'$ such that $N = \langle P_c(S), A, Proj, U\rangle$, $D' = \langle P_c^{'}(S), A, Proj^{'}, U\rangle$. The special case described above obtains is when the probability distribution $P_c^{'}$, as well as every projected distribution $P_i^{'}$ returned by the projection function $Proj^{'}$ are deterministic.

4 Conclusions and Future Work

In this paper, we intended to show how two diverse strands of work in AI and cognitive science can be combined to create a fundamental theory of emotions and personality. We started from the principled paradigm of rational agent design based on decision theory. We then used this paradigm to formally define the emotional states and personality of an artificial intelligent agent. We view the emotional states as the agent's decision-making modes, predisposing the agent to make its choices in a specific, yet rational, way. Change of the emotional state, say due to an external stimulus, invokes a transformation of the agent's decision-making behavior. We defined personality as consisting of the agent's emotional states and specification of transitions taking place among the states. We also defined a notion of a personality model, which is a probabilistic and learnable version of the personality construct.

Having the formal definitions of personality and emotions allows us to show how, and why, they are useful to a rational artificial agent. First, by modifying the decision-theoretic model used for deliberative rationality, emotions allow the agent to control the allocation of its cognitive resources and the complexity of its deliberations under time pressure in an uncertain environment. For example, limiting the number of alternative behaviors considered, or shortening the time horizon of these alternatives allows for more rapid decision-making. Second, emotions and terms related to personality are valuable when the agent finds it useful to inform other agent(s) about its own internal state. Instead of having to describe the details of its internal state, and running the risk of being misunderstood if the other agents are engineered differently, the agent can use more abstract and universal terms. For example, notions of stress or panic may be convenient to express the fact that the urgency of the situation forced the agent to look at only short-term effects of its actions. Or, say, anger, may be a good way to summarize the agent's belief that only hostile behavioral alternatives are worth considering and putting into action. Thus, in the context of communication with other agents, the formal definitions of emotions provide semantics of terms with which the agents can express their own internal states and understand the states the other agents are in. Third, well-defined emotional states of self and others are crucial in the agent's interaction with humans. Frequently, human-computer interaction is impeded by the machine being hopelessly out-of-step with the emotional state of the human user. However, since the user's emotional state can be now assessed using measurable and inferred factors, it is important for the machine to understand the user's emotional state and model its effects on the user's decision-making and his/her tendency to action.

The result of our work is a fundamental and principled understanding of the notions of personality and emotions in artificial intelligent agents. This, we believe, will significantly enhance the applications of intelligent systems in dynamic and unpredictable environments, multi-agent systems, and in human-computer interaction.

Our future work will provide clearer definitions of emotional states as decision-making modes, use these definitions to design useful personalities of agents operating in various environments, and to investigate robust methods of learning of personality models used during interaction and communication with other agents. We will investigate the effects of various modifications of the basic decision-making model, as defined above, on the abilities of the agents to accomplish goals under conditions of emergency and time pressure. Further, we will go beyond the simple transformations exemplified above, and create a comprehensive catalogue of transformations and their mixtures. We will experiment with agents implemented in the simulated environments of Tile-World, Wumpus Environment, and MICE. The agents will be made to satisfy given goals but act under time pressure and face emergency situations.

Also needed are extensive experiments with agents endowed with simple personalities, acting under moderately variable simulated environments. We will concentrate on agents under time pressure and attempt to determine what emotional states and decision-making modes are most useful, given parameters of a moderately variable environment. We expect that more complex personalities, with more five and more emotional states, will be needed to deal with complex environments, calling for the agents to be, for example, curious in their explorations, but also ready to go into panic mode under time pressure, and exhibiting angry and aggressive behavior when threatened.

Acknowledgements

This work has been supported by the Office of Naval Research Artificial Intelligence Program under contract N00014 -95-1-0775, and by the National Science Foundation CAREER award IRI-9702132.

References

1. Robert Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.
2. Cristina Bicchieri. *Rationality and Coordination*. Cambridge University Press, 1993.
3. Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial intelligence Research*, 11:1–94, 1999.
4. David Carmel and Shaul Markovitch. Learning models of intelligent agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 62–67, Portland, OR, August 1996.
5. P. R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.
6. G. Cottrell and Metcalfe. Empath: Face, emotion and gender recognition using holons. In *Advances in Neural Information Processing*. Morgan Kaufman Publishers, 1991.
7. D. Dennett. Intentional systems. In D. Dennett, editor, *Brainstorms*. MIT Press, 1986.
8. Antonio R. Dimasio. *Descartes' Error*. Grosset/Putnam, 1994.
9. Jon Doyle. Rationality and its role in reasoning. *Computational Intelligence*, 8:376–409, 1992.
10. Edmund H. Durfee, Jaeho Lee, and Piotr Gmytrasiewicz. Overeager rationality and mixed strategy equilibria. In *Proceedings of the National Conference on Artificial Intelligence*, July 1993.
11. N. H. Fridja. *The Emotions*. Cambridge University Press, 1986.

12. Piotr J. Gmytrasiewicz and Edmund H. Durfee. Rational coordination in multi-agent environments. *Autonomous Agents and Multiagent Systems Journal*, 3(4):319–350, 2000.
13. Piotr J. Gmytrasiewicz and Edmund H. Durfee. Rational communication in multi-agent environments. *Autonomous Agents and Multiagent Systems Journal*, 4:233–272, 2001.
14. B. Hayes-Roth, B. Ball, C. Lisetti, and R. Picard. Panel on affect and emotion in the user interface. In *Proceedings of the 1998 International Conference on Intelligent User Interfaces*, pages 91–94, 1998.
15. W. James. What is an Emotion? *Mind*, 9:188–205, 1884.
16. W. James. The Physical Basis of Emotion. *Psychological Review*, 1:516–529, 1894.
17. P. N. Johnson-Laird and K. Oatley. Basic Emotions, Rationality, and Folk Theory. *Cognition and Emotion*, 6(3/4):201–223, 1992.
18. S. Kraus and K. Sycara. Argumentation in negotiation: A formal model and implementation. *Artificial Intelligence*, 104(1-2):1–69, 1989.
19. Victor Lesser, Michael Atighetchi, Brett Benyo, Raja Bryan Horling, Vincent Anita, Wagner Regis, Ping Thomas, Shelley Xuan, and ZQ Zhang. The intelligent home testbed. In *Proceedings of the Autonomy Control Software Workshop (Autonomous Agent Workshop)*, 1999.
20. H. Leventhal and K. Scherer. The relationship of emotion to cognition:a functional approach to semantic controversy. *Cognition and Emotion*, 1(1):3 – 28, 1987.
21. Christine L. Lisetti and David E. Rumelhart. Facial expression recognition using a neural network. In *Proceedings of the 1998 Florida Artificial Intelligence Research Symposium (FLAIRS'98)*, 1998.
22. Christine L. Lisetti and Diane J. Schiano. Automatic facial expression interpretation: Where human interaction, artificial intelligence and cognitive science intersect. *Pragmatics and Cognition, Special Issue on Facial Information Precessing and Multidisciplinary Perpective*, 1999.
23. M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 1994.
24. Barry O'Neill. Approaches to modelling emotions in game theory. Technical report, Department of Political Science, Stanford University, http://www.stanford.edu/∼boneill/emotions.html, 2000.
25. Andrew Ortony, Gerald Clore, and Allen Collins. *Cognitive Structure of Emotions*. Cambridge University Press, 1988.
26. R. Picard. *Affective Computing*. MIT Press, 1997.
27. Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter*. MIT Press, 1994.
28. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.
29. Tuomas Sandholm and R. H. Crites. Multiagent reinforcement learning and iterated prisoner's dilemma. *Biosystems Journal*, 37:147–166, 1995.
30. H. Simon. Motivational and Emotional Controls of Cognition. *Psychological Review*, 74:29–39, 1967.
31. A. Sloman. Motives, Mechanisms, and Emotions. In M. Boden, editor, *The Philosophy of Artificial Intelligence*. Oxford: Oxford University Press, 1990.
32. A. Sloman and M. Croucher. Why robots will have emotions. In *Proceedings of the Seventh IJCAI Vancouver, B.C.*, pages 197–202. San Mateo, CA: Morgan-Kaufmann, 1981.
33. K. Sycara. Multiagent systems. *AI Magazine*, 10(2):79–93, 1998.
34. Michael Wooldridge and Editors Anand Rao. *Foundations of Rational Agency*. Kluwer Academin Publishers, 1999.

The Ψ Calculus: An Algebraic Agent Language

David Kinny

Intelligent Agent Laboratory, Department of Information Systems
University of Melbourne, 3010 Australia
and
AWARE RESEARCH Pty. Ltd.

Abstract. Ψ is a novel algebraic language for the specification of agents, such as BDI agents, which employ a sense–compute–act computation cycle and stored plan execution as the basis of agent behaviour. It generalizes and extends agent architectures such as PRS and dMARS in several ways, and possesses a complete operational semantics covering all aspects of agent computation from intention step execution to the top-level control cycle. This is specified uniformly in process algebraic style by rewrite rules, and has certain safety, guarantee and compositionality properties which facilitate reasoning about agent program behaviour.

1 Introduction

In the decade since Shoham proposed the notion of Agent-Oriented Programming [19], there has been a remarkable explosion of interest in software agents and of activities focused on their application. Numerous proposals for agent architectures of various types have appeared [14] and many have been at least partially implemented, but remarkably few true agent programming languages have been developed. By this we mean languages which provide relevant abstractions, e.g., goals, plans and actions, and have appropriate, well-developed computational models which, taken together, simplify and guide the tasks of constructing reliable agents and multi-agent systems and ensuring that they behave in accordance with their specifications. The mainstream trend today seems to be to build agents in Object-Oriented (OO) languages such as Java, painstakingly constructing *ad hoc* solutions with toolkits which supply infrastructure more than architecture and little resembling a coherent agent-oriented programming framework.

An OO approach to building agents can undoubtedly leverage a rich and mature programming framework, but is arguably like doing OO programming in C, as OO languages lack appropriate agent abstractions, forcing one to construct by hand things one should get for free, although the use of design patterns can provide some relief. What they most lack, however, are computational models that effectively support the concurrent internal activities needed in all but the simplest agent designs, e.g., perception and belief update, reactive and proactive behaviour generation, and external action. The responsibility to get it right is shifted to the programmer, with a consequent increase in development cost and a loss of transparency, reliability and potential for reuse.

An obvious and well-understood approach to addressing such deficits is to define a high-level agent language, and construct an interpreter which implements it or a compiler for it that generates code in an object-oriented target language. This is exactly the approach adopted by the well-known agent technologies PRS [4] and dMARS [7], which

J.-J.Ch. Meyer, M. Tambe (Eds.): Intelligent Agents VIII, LNAI 2333, pp. 32–50, 2002.

are implemented by interpretation in Lisp or compilation to C++ respectively. Both employ architectures based on reactive and goal-directed execution of stored plans, and provide a rich, graphical plan language as the primary means for the specification of agent behaviour. Other true agent languages include those based on rule execution in a logic programming framework, such as 3APL [6], ConGolog [10] and Vivid [20].

A common failing of true agent languages, however, is that they lack an adequate formal semantics, which is essential for understanding and exact reasoning about agent program behaviour. This is particularly true of plan-based languages, whose execution semantics is intrinsically complex. The essential reason for this is that plans, unlike rules, are executed step by step so that any given point there may be many active plan instances (usually called *intentions*) whose execution must somehow be coordinated. Indeed, there may be several active instances of a single plan. Concurrency is inevitable, and a surfeit of choices about how exactly it should be controlled is available, with different sets of choices resulting in subtly or radically different execution semantics. A further architectural design choice is to what extent the agent language should impose a particular computational model or offer a selection of them, and how to provide means by which agent programmers can influence or determine how execution proceeds.

Despite the prominence of plan execution systems such as PRS and dMARS, there has been to date no systematic study of what control choices are available and appropriate, and of their consequences for agent language semantics. The historical trend has been to define mechanisms and build implementations whose exact semantics then arises rather opaquely from the particular choices made. The semantic vagueness of such systems has been long lamented, and attempts to reverse engineer a formal semantics have been made [3,17], but these been simplified treatments which fail to capture lucidly all the complexities of plan execution. A more logical approach is first to define an agent model and language, choosing an appropriate semantics for its constructs, and then produce an implementation and verify its conformance to the intended semantics.

In recent work [8] we have pursued the latter approach, making a detailed study of control in plan execution systems and its semantic consequences. One outcome has been the development of a novel algebraic language – the Ψ calculus – for the representation of agents and their internal elements. Ψ provides two things: a generic *agent model*, one element of which is an abstract plan-based programming language which generalizes and extends in significant ways those of PRS and dMARS; second, a means for directly specifying its operational semantics. The latter is captured uniformly in process algebraic style by a hierarchy of *reduction systems*, and covers all aspects of agent computation including high-level control policies that are usually specified indirectly by algorithms and opaque selection functions [17], or expressed in a distinct meta-language [6].

Ψ offers a very flexible framework in which the effects on language semantics of key design and control decisions may be explored. Our analysis of these issues leads to the conclusion that many shortcomings and limitations of existing plan execution systems are the direct consequence of inadequate policies or mechanisms for control of the many sources of non-determinism which can arise. Fortunately, a third result has been to identify a novel, intuitive *standard semantics*, based on a well-defined computation property called *reliability*, which addresses these defects and ensures desirable safety, guarantee and compositionality properties that facilitate reasoning about agent program behaviour.

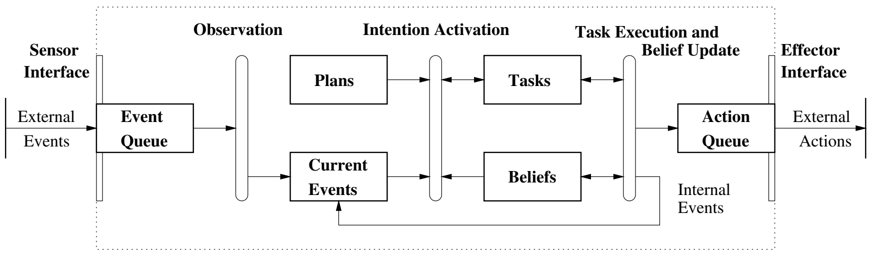


Fig. 1. The Abstract Ψ Agent Model

The presentation here of Ψ is introductory, limited in scope, and focuses on what exactly constitutes a Ψ agent and how the operational semantics is given. It begins by presenting the agent model and key assumptions about an agent's relationship with its environment, and then describes the representation of a Ψ agent's internal structure and informally introduces the semantics of the intention language in which behaviour is expressed. In Section 3 we introduce reduction systems and formalize the key elements of the operational semantics of Ψ computation, demonstrating how different computational models may be captured, including that of the standard semantics. We conclude with a brief summary. The interested reader may find elsewhere definitions of reliability and the standard semantics, and a treatment of language features not covered here and other issues such as the relationship of Ψ to other languages and formalisms [8,9].

2 The Ψ Agent Model

A Ψ (PSI) agent is an encapsulation of state and behaviour whose internal elements include *beliefs* (an epistemic component) and *intentions* (a procedural component), the latter including *tasks* and *plans*. (PSI arose as an acronym for Plan System Interpreter.) It interacts with its environment via interfaces of two types: *sensors* from which the agent receives events that carry information from its environment, and *effectors* or actuators by which the agent performs actions that are intended to affect its environment. These will be called *external events* and *external actions* to distinguish them from *internal events* which may be generated as a result of changes in the agent's state, of which *goals* are a special case, and from *internal actions* which cause such changes.

Such an agent is depicted in Figure 1. Components of the agent's state are denoted by rectangles, and its computational processes by narrow ovals. The state components are structured aggregations (sequences, sets, multisets, etc.) of elements such as events, actions, beliefs, plans and tasks. A Ψ agent's state is represented formally by an *agent configuration* $\psi = \langle \mathcal{Q}_{\mathsf{E}}, \xi, \beta, \omega, \delta, \mathcal{Q}_{\mathsf{A}} \rangle$ whose elements are respectively an *event queue*, a *signal state*, a *belief state*, an *intention state*, a *response state* and an *action queue*. The set of possible states of an agent is just its *configuration space* Ψ – the direct product of the state spaces of these elements. In the general case, when there are no restrictions on the size of queues or the number of intentions in ω, the configuration space is infinite; this is a feature which distinguishes Ψ from frameworks such as reactive systems [11]. An *agent computation* is defined as usual as a sequence $\psi_0 \longrightarrow_1 \psi_1 \longrightarrow_2 \cdots$ of transitions between agent configurations, and may be captured by a *computation relation*.

2.1 External Interactions

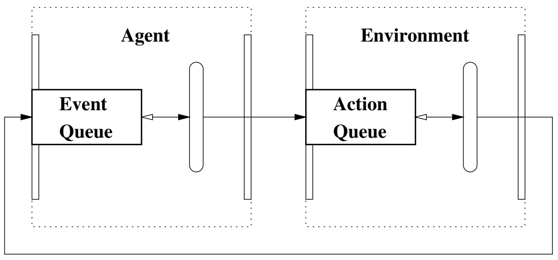
A Ψ agent's event and action queues are the interfaces between it and its environment, which is taken to include its sensors and effectors. Events generated by sensors, which provide information about the state of its environment, and the actions it takes to affect that environment are represented by terms of its *external event languages* $\mathcal{L}_{\mathsf{E}_e} \subset \mathcal{L}_{\mathsf{E}}$ and $\mathcal{L}_{\mathsf{A}_e} \subset \mathcal{L}_{\mathsf{A}}$. Individual event or action terms $\mathsf{e} = e(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ and $\mathsf{a} = a(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ consist of a symbol which identifies the *event type* applied to a sequence of value terms which constitute its *properties*. They thus resemble atoms of a first-order logic, but will not be interpreted logically. While that is one possibility, another is as messages in an OO sense or as messages of an agent communication language. The representation captures a minimal set of structural requirements without any semantic baggage.

The event languages allow logical variables, denoted $\mathrm{x}, \mathrm{y}, \mathrm{x}_1, \ldots,$ inside terms. Ground terms, denoted $\underline{\mathsf{e}}, \underline{\mathsf{a}}$, stand for events and actions, whereas non-ground terms are patterns which can match distinct instances of a particular event or action type. We use E ($\underline{\mathsf{E}}$) and A ($\underline{\mathsf{A}}$) to denote (ground) *compound events*: sets of events and action terms. In general, if T denotes a compound term, $\dot{\mathsf{T}}$ is a non-empty one, $\ddot{\mathsf{T}}$ a non-singleton one, $\star$ an empty one, $\underline{\mathsf{T}}$ a ground one, and $\mathrm{var}(\mathsf{T})$ is the set of logical variables it contains.

Design decisions about interaction determine the expressiveness of an agent model. In Ψ perception is modelled as an external process which inserts ground external events into the event queue $\mathcal{Q}_{\mathsf{E}}$. We assume in the general case it is uncontrollable: that sensors create events at unpredictable time points and asynchronously insert them into the event queue, and do not assume that events carry properties such as timestamps. To enable the preservation of temporal ordering relationships amongst events including simultaneity, the event queue is modelled as a sequence of compound events, and perception as the concatenation $\otimes$ of a set of events $\dot{\mathsf{E}}$ to the end of the queue. As the events in a set are necessarily distinct, an event cannot occur more than once at the same instant. A strict total ordering over events corresponds to the case where each such set is a singleton.

Action is under an agent's control, in the sense that it may initiate an action when it chooses, but the effects of an external action are not; they are determined by its effectors and its environment, and may be immediate or delayed. In Ψ, an agent's effectors, like its sensors, are modelled as an uncontrolled external process, and its action queue as a sequence of compound external actions $\dot{\mathsf{A}}$. Just as the perception process is a producer of events which are consumed by the agent, the external action process is symmetrically a consumer of actions which are produced by the agent. The agent completely controls its consumption of events and its production of actions, but not these external processes.

Abstracting from its internal components and operations, a Ψ agent may thus be characterized as a process that consumes external events and produces external actions that are produced and consumed by perception and action processes in its environment. Its coupling with its environment is buffered and asynchronous, and so an agent and its environment can be represented symmetrically as a system of directly coupled Ψ agents, by identifying the agent's event queue with the environment's action queue, and *vice versa*. Thus an embedded agent can also be viewed by an observer as a single reduced configuration $\langle \mathcal{Q}_{\mathsf{E}}, \mathcal{Q}_{\mathsf{A}} \rangle$ and two transitions, one representing the observable behaviour of the agent, and the other the observable behaviour of its environment, as in Figure 2. The use of such transition rules to represent behaviour will be explained in Section 3.



$$\frac{\dot{\mathsf{A}} \subset \mathcal{L}_{\mathsf{A_e}}}{\langle \mathcal{Q}_{\mathsf{E}}, \mathcal{Q}_{\mathsf{A}}\rangle \longrightarrow \langle \mathcal{Q}'_{\mathsf{E}}, \mathcal{Q}_{\mathsf{A}} \otimes \langle \dot{\mathsf{A}}\rangle\rangle}$$

$$\frac{\dot{\mathsf{E}} \subset \mathcal{L}_{\mathsf{E_e}}}{\langle \mathcal{Q}_{\mathsf{E}}, \mathcal{Q}_{\mathsf{A}}\rangle \longrightarrow \langle \mathcal{Q}_{\mathsf{E}} \otimes \langle \dot{\mathsf{E}}\rangle, \mathcal{Q}'_{\mathsf{A}}\rangle}$$

Fig. 2. A Ψ Agent embedded in an Environment

2.2 Representation of Internal Components

Beliefs. A Ψ agent needs beliefs because the events its sensors produce may carry only a quantum of information about some aspect or part of or change in its environment rather than a complete model of it, and because it is computationally less expensive to maintain a cached model than recompute it from scratch every time something changes. Conceptually, we regard an agent's beliefs as a database, shared by its tasks, supporting basic query and update operations. For our purposes, the details of how beliefs are represented and their exact properties are unimportant, they are viewed quite abstractly, without any commitment to their structure, logical properties, veracity or consistency; we merely assume a set $\mathcal{B}$ of belief states $\beta, \zeta, \beta_1, \ldots$, called a *belief space*, and represent queries and updates by transition relations on $\mathcal{B}$, called *belief transition relations*. These will be the building blocks from which the operational semantics is constructed.

Definition 1 (*Transition relation, Condition, Update*).

- $\diamond$ A *transition relation* on a set of states S is a binary relation $\rightarrow\ \subseteq S \times S$.
- $\diamond$ The *identity relation* $=$ is defined as $\{\,(s, s) : s \in S\,\}$.
- $\diamond$ A transition relation $\rightarrow$ is a *condition* if $\rightarrow\ \subset\ =$, and an *update* if $\rightarrow\ \not\subseteq\ =$.

We write transition relations infix, using $r \rightarrow s$ for $(r, s) \in \rightarrow$, $r \rightarrow$ for $\exists\, s : r \rightarrow s$, $r \not\rightarrow$ for $\neg(r \rightarrow)$, and $\rightarrow^{=}$ to denote a reflexive closure ($\rightarrow \cup =$). We will say that a transition $\rightarrow$ is *enabled* in s if $s \rightarrow$ and *vacuous* in s if $s \rightarrow s$ and $s \rightarrow r \Rightarrow r = s$.

To facilitate examples, beliefs will also be represented concretely as a finite set of untyped state variables $\mathrm{u}, \mathrm{u}_1, \ldots$ ranging over a data domain containing booleans, integers, and structured values such as lists and sets. Belief states can then be captured by valuations of these variables in the conventional manner, and their values in particular states will be denoted by $\beta[\![\mathrm{u}]\!]$, or when the state is obvious from the context, by $[\![\mathrm{u}]\!]$.

Queries. Queries are captured by a condition language $\mathcal{L}_{\mathsf{C}}$ whose elements are *atomic conditions* $\mathsf{c}, \underline{\mathsf{c}}, \mathsf{c}_1, \ldots$. Like events these terms may contain logical variables, allowing output values and non-deterministic conditions such as set membership. Their semantics is given by an interpretation function $[\![\_]\!]$ which maps ground conditions $\underline{\mathsf{c}} \in \mathcal{L}_{\mathsf{C}}$ to boolean functions on belief states, whose application to a state β will be written postfix as $\beta[\![\mathsf{c}]\!]$. *Compound conditions* $\mathsf{C}, \underline{\mathsf{C}}, \mathsf{C}_1, \ldots$ are sets of these, denoting conjunctions: the belief transition relation $\xrightarrow{\mathsf{C}}$ induced by a condition C is $\{\,(\beta, \beta) : \forall\, \mathsf{c}_i \in \mathsf{C}, \beta[\![\mathsf{c}_i]\!]\,\}$, and is equivalent to the intersection of the individual relations $\xrightarrow{\mathsf{c}_i}$.

Updates. The *epistemic effect* of an event or action upon an agent's belief state is similarly given by an interpretation function $[\![\_]\!]$ which associates a belief transition relation with each event $\mathsf{e} \in \mathcal{L}_\mathsf{E} \cup \mathcal{L}_\mathsf{A}$. If an event has no effect the relation is the identity $= (\xrightarrow{\emptyset})$, and in the case of an illegal or non-ground term it is the null relation $\emptyset$. We will write $\beta \xrightarrow{\mathsf{e}} \zeta$ for $\beta[\![\mathsf{e}]\!]\zeta$, and $\beta \xrightarrow{\mathsf{e}}$ for $\exists\zeta : \beta \xrightarrow{\mathsf{e}} \zeta$. The epistemic effects of compound events $\xrightarrow{\mathsf{E}}$ and actions $\xrightarrow{\mathsf{A}}$ are defined by composition of the effects of their elements, but only if they *commute*, i.e., their net effect is independent of the order of composition. If not, the resulting relation is null, and the event or action is *not executable*.

Basic transitions. The fundamental construct upon which the Ψ plan language is based is a triggerable, guarded, compound action, referred to as a *basic transition*.

Definition 2 (*Basic transition*).

Let $\mathcal{L}_\mathsf{E}$, $\mathcal{L}_\mathsf{C}$, and $\mathcal{L}_\mathsf{A}$ be respectively an agent's event, condition and action languages. A *basic transition* $\mathcal{X} = \langle \mathsf{E}, \mathsf{C}, \mathsf{A} \rangle$ is a tuple of three components:

– a finite set $\mathsf{E} \subset \mathcal{L}_\mathsf{E}$ of events, called the *trigger*,
– a finite set $\mathsf{C} \subset \mathcal{L}_\mathsf{C}$ of atomic conditions, called the *guard*, and
– a finite set $\mathsf{A} \subset \mathcal{L}_\mathsf{A}$ of actions, simply called the *action*.

A basic transition is *complex* if $\mathsf{E} \neq \emptyset$ else it is *simple*, and is *guarded* if $\mathsf{C} \neq \emptyset$ else it is *unguarded*. If complex or guarded it is *restricted*, and if simple and unguarded it is *free*.

Basic transitions, recognizable as elements of formalisms such as Statecharts [5] and Active Databases [21], will be denoted by $\xrightarrow{\mathsf{E[C]A}}$ or by ECA. Informally, the *standard semantics* of transition execution is that if the events in E simultaneously occur when the guard condition C holds then the actions in A *must* be performed, atomically. Otherwise, $\mathcal{X}$ is disabled. Any part may be empty: if E is empty the action must be performed when C holds, an empty guard $\star$ holds vacuously, and an empty action $\star$ has no effect.

Signals. Transition execution is defined in terms of the *signalling* of events and the transition relations associated with guard conditions and actions. An agent's *signal state* ξ is a set of ground "current" events, called *signals* and distinguished as S rather than E. An event *is signalled* by becoming a member of the signal state, either directly in the case of internal events, or as a result of an *observation* step, which takes external events from the event queue, performs any epistemic effects they have, and adds them to ξ.

Execution of an enabled transition can consist of up to three stages: *activation*, *liberation*, and *action execution*. Activation, in the basic case, is just simplification: pattern matching between subsets of the signal state and the trigger of a *simplifiable* transition ($\exists\theta : \dot{\mathsf{E}}\theta \subseteq \xi$), eliminating the trigger, and application of the substitution θ that results from the matching to the guard and action. A *satisfiable* guarded transition ($\exists\theta : \beta \xrightarrow{\dot{\mathsf{C}}\theta}$) is liberated by finding a (minimal) substitution θ satisfying the guard, eliminating the guard, and applying θ to the action. Executing the action of an executable free transition ($\beta \xrightarrow{\mathsf{A}} \zeta$) consists in performing its epistemic effects, queueing any external actions $\mathsf{A} \cap \mathcal{L}_{\mathsf{A}_\mathsf{e}}$ it contains, signalling any internal events $\mathsf{A} \cap \mathcal{L}_{\mathsf{E}_\mathsf{i}}$, and eliminating the transition.

Responses. Why should one discriminate between these stages rather than just describe an integrated transition execution process that combines them? The answer is that to model concurrent or overlapped transition execution by interleaving, one must adopt a suitably fine-grained model of individual transition execution. For example, if several transitions are enabled in a given state, it may be desirable to activate them all before performing the actions of any, or to perform their combined actions atomically. An agent's *response state* δ is a compound action which permits control policies that execute multiple transitions in one computation step by accumulating their individual actions and deferring their effects until all have been selected and executed.

Intentions. The *plan language* $\mathcal{L}_P$ in which behaviour is specified is part of a larger *intention language* $\mathcal{L}_\Omega$ whose elements include plans, tasks and processes representing multisets of these. All processes are constructed from basic transitions and the inactive *null process* 0, denoting successful termination, by means of the 4 process combinators:

$\cdot$ *prefixing*, which joins a transition $\mathcal{X}$ and any process π into a *unary task* $\tau = \mathcal{X} \cdot \pi$,
$!$ *replication*, which transforms a unary task τ into a *plan* $\rho = !\,\tau$,
$+$ *choice*, which combines any tasks σ_1 and σ_2 into a *branching task* $\sigma = \sigma_1 + \sigma_2$, and
$\times$ *parallel*, which combines any processes π_1 and π_2 into a *multiprocess* $\pi = \pi_1 \times \pi_2$.

As well as using ω and π to denote any process, ρ a plan, and σ an arbitrary task, of which a unary task τ and 0 are special cases, we use ϕ to denote an intention (ρ or σ).

Prefixing is a restricted, asymmetrical form of sequential composition. Replication allows a task to be executed an unbounded number of times without being consumed; normally this occurs only in response to activation by a signal, i.e., the prefix of a plan, called its *activation transition*, is usually a complex transition. We will omit parentheses if possible, taking the operator precedence order to be $\times < + < \, ! < \cdot < \langle$*juxtaposition*$\rangle$.

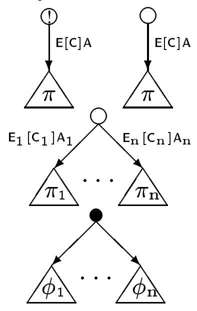
Unlike $\cdot$, the operators $+$ and $\times$ are commutative and associative, so one can omit parentheses from nested sums or products, and write any task σ as a sum of unary tasks $\tau_1 + \cdots \tau_n = \sum_{i=1}^{n\geqslant 0} \tau_i$, and any multiprocess π as a product of intentions $\phi_1 \times \cdots \phi_n = \prod_{i=1}^{n\geqslant 0} \phi_i$. The null process 0 is included, as by definition $0 \overset{\diamond}{=} \sum_{i=1}^{0} \tau_i \overset{\diamond}{=} \prod_{i=1}^{0} \phi_i$. The algebraic structure is that a multiprocess π is a multiset of concurrent behaviours, and a task σ is a set of alternative behaviours, each one captured by a prefixed process. A unary task τ consists of just one such alternative, and the null process 0 of none.

Plans and tasks have a natural graphical representation as finite trees. Each transition corresponds to an arc, and each combinator to a node to which the subtrees corresponding to the components are adjoined. The correspondence is formalized as follows.

Definition 3 (*Tree representation of intentions*).

Let $\omega \in \mathcal{L}_\Omega$ be any process. The corresponding tree is defined inductively as follows.

- ◇ A plan $\rho = !\mathcal{X} \cdot \pi$ is represented by an arc labelled with the transition $\mathcal{X}$ from a replication node $\textcircled{!}$ to a tree corresponding to π. A unary task $\tau = \mathcal{X} \cdot \pi$ is identical except for its root ○.
- ◇ A branching task $\ddot{\sigma} = \sum_{i=1}^{n>1} \mathcal{X}_i \cdot \pi_i$ is represented by joining the roots of the trees corresponding to the summands. An empty summation 0 is represented by a node ○ with no outgoing arcs.
- ◇ A compound multiprocess $\ddot{\pi} = \prod_{i=1}^{n>1} \phi_i$ is represented by a fork node ● with outgoing unlabelled arcs to the roots of each of the trees corresponding to the component intentions.



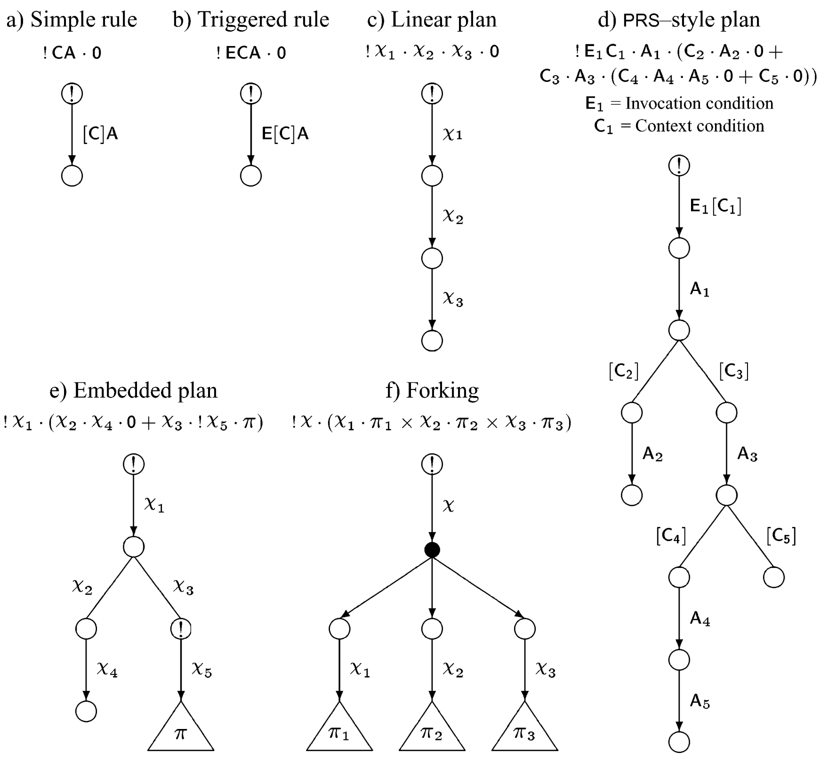


Fig. 3. Plans as Trees

Examples of types of plans and the corresponding trees appear in Figure 3. A fork node ● is a point where execution cannot pause. The activation transition of a PRS-style plan cannot contain an action, and all its other transitions are either guards or actions.

Agents. Having fixed a belief space $\mathcal{B}$, event, condition and action languages $\mathcal{L}_\mathsf{E}$, $\mathcal{L}_\mathsf{C}$, and $\mathcal{L}_\mathsf{A}$, and so an intention language $\mathcal{L}_\Omega$ and an agent configuration space $\varPsi$, a specific agent is defined by an initial belief state $\beta_0 \in \mathcal{B}$ and an initial intention state $\omega_0 \in \mathcal{L}_\Omega$, which determine its *initial configuration* $\psi_0 = \langle \star, \star, \beta_0, \omega_0, \star, \star \rangle \in \varPsi$. Possible agent computations may then be captured by defining a specific computation relation on $\varPsi$.

2.3 Informal Semantics of Intentions

To conclude this section we introduce the semantics of intention execution in Ψ, to be formalized in the sequel. The simplest interesting process is the unary task $\mathcal{X} \cdot 0$, whose execution is equivalent to the transition $\mathcal{X}$, since 0 represents successful termination. We will call such a process a *rule*, and a nested prefixed process $\mathcal{X}_1 \cdot \ldots \cdot \mathcal{X}_n \cdot 0$, a *linear task*. Its execution consists in executing first its *head* $\mathcal{X}_1$ and then the remainder, its *body*, transition by transition. Any substitution applied during the execution of its head is also applied to its body. Exactly the same applies to a unary task $\mathcal{X} \cdot \pi$ with an arbitrary body. Intuitively, a branching task $\ddot{\sigma} = \tau + \dot{\sigma}$ executes by choosing one of its *branches* τ

to execute and pruning all the others $\dot{\sigma}$. If many are enabled then one is chosen non-deterministically. It should be clear why disjunction is not needed in guard conditions, since $\mathsf{E}[\mathsf{C}_1]\mathsf{A} \cdot \pi + \mathsf{E}[\mathsf{C}_2]\mathsf{A} \cdot \pi$ implements $\mathsf{E}[\mathsf{C}_1 \vee \mathsf{C}_2]\mathsf{A} \cdot \pi$. Choice may also be used to implement more powerful forms of disjunction, such as $\mathcal{X}_1 \cdot \pi + \mathcal{X}_2 \cdot \pi$.

In the simplest case, a multiprocess or *compound intention* $\ddot{\pi} = \phi \times \dot{\pi}$ executes by choosing some enabled intention ϕ and executing it, retaining all others unaffected. When many intentions are enabled some sort of intention selection strategy is required, and the simplest policy is to choose one non-deterministically. While widespread among agent languages, this policy has many drawbacks [8], not the least of which is that it is not *fair:* some enabled intention might never be chosen if it always has competitors.

Conceptually, a plan $\rho = \,!\,\tau$ cannot be executed directly, only activated. Technically, its execution consists in replicating instances of it (*sans* replication operator) and executing them, i.e., the creation of one or more new tasks. A key semantic design decision is how this should be done when a plan can be activated in several distinct ways.

A task may contain a parallel or replication operator in its body. In the former case (Figure 3(f)), when it reduces to this point it forks into independent parallel tasks, and in the latter (Figure 3(e)) a new plan is introduced. This does not reflect a general capacity for planning, as such a plan can only specialize a subtree of an existing plan.

Process execution can be captured more formally by *reduction rules*, such as:

$$\frac{\mathcal{X} \xrightarrow{\theta,\alpha} \mathsf{0}}{\mathcal{X} \cdot \pi \xrightarrow{\alpha} \pi\theta} \qquad \frac{\tau \xrightarrow{\alpha} \pi}{\tau + \sigma \xrightarrow{\alpha} \pi} \qquad \frac{\phi \xrightarrow{\alpha} \pi}{\phi \times \omega \xrightarrow{\alpha} \pi \times \omega} \qquad \frac{\tau \xrightarrow{\alpha} \pi}{!\,\tau \xrightarrow{\alpha} \,!\,\tau \times \pi}$$

These sequents capture the *contexts* in which process execution can occur, the structural changes it causes, and the actions that result. For example, the first rule says that if a transition $\mathcal{X}$ can execute via a substitution θ and perform an action α, then the unary task $\mathcal{X} \cdot \pi$ can execute and perform α to become $\pi\theta$. Note how plan replication is concisely defined in terms of unary task execution by means of the parallel composition operator.

The reduction approach to representing process execution also has the advantage that it dispenses with the notion of a locus of control and a local environment. A task's locus of control is implicit – always its head – and prefixes (and branches not chosen) are eliminated as it executes. While a task is indeed an instance of a plan which can be thought of as possessing local state in the form of single-assignment logical variables, these are bound as the task reduces, i.e., it is progressively specialized as it undergoes transformations permitted by the reduction rules. That the state is local is a consequence of the execution semantics of multiprocesses: substitutions are applied only within a task. Parallel processes do, however, share global state, captured by the agent's beliefs.

The intention language $\mathcal{L}_\Omega$ is quite expressive when compared to other agent programming languages, providing triggered guarded action as a primitive, and so allowing a task to wait during its execution not just for conditions but also for events. Many other agent languages can be expressed as restricted, special cases. It is similar in basic structure to Milner's classic π-calculus [13]. The main differences are the absence of names and input/output actions. Signals may be viewed as playing a rôle similar to input/output actions, however the crucial difference is that whereas input and output actions in the π-calculus allow synchronized communication between pairs of processes, signals in Ψ allow broadcast communication between all intentions, i.e., a single signal can trigger many transitions. In this respect Ψ resembles Prasad's broadcast calculus CBS [16].

3 Operational Semantics of Ψ Computation

Having defined the structure of a Ψ agent, the task at hand is to give an operational semantics not just for its intention language, but for all aspects of agent computation, i.e., to choose and define an agent computation relation. The intention language $\mathcal{L}_\Omega$ has a *naïve semantics* as described and captured by the reduction rules above, but basic transitions can be given a variety of different semantic treatments, and further choices arise when activation and execution of parallel intentions is considered, and again when perception and external action is introduced. The primary challenge is twofold: to make sensible choices, and to capture these in a complete, compact and comprehensible way.

The naïve semantics, which arises from treating basic transition execution as atomic and allowing uncontrolled interleaving of parallel intentions, is seriously problematic and unintuitive in several respects. For example, an intention waiting for a signal may not be activated by it, a plan which should be activated more than once by distinct signals may be activated only once if they happen to occur concurrently, and an enabled transition may be arbitrarily delayed and even become disabled before it can execute. These and other problems occur because non-determinism that arises during execution is resolved uniformly by making an arbitrary choice from the possibilities available. Intention activation and execution are *inherently unreliable* – permitted but not required to occur – and as a result agent programs are non-compositional: adding a new plan to an agent may unpredictably, even radically, change the behaviour of its existing plans.

Such problems occur to some extent in most existing true agent languages; to avoid them and achieve a useful and intuitive semantics one must impose control upon the various sources of non-determinism which can arise, making specific, consistent choices about what to do and when to do it. There are many ways in which this may be done, and the particular way which constitutes the *standard semantics* of Ψ is to adopt a synchronous broadcast semantics for signals, a *must* rather than a *may* semantics for basic transition execution, and a *step semantics* for parallel intentions which ensures that all execute on an equal footing. Together, these constitute a strong fairness principle, called *reliability*, which goes beyond existing notions of fairness such as *justice* and *compassion* [11] and ensures that intention activation and execution has certain guarantee and compositionality properties which facilitate reasoning about agent program behaviour.

We have demonstrated elsewhere [8,9] that reliable execution cannot be achieved for arbitrary agent intention states, but it is possible to identify precisely the conditions under which it can and cannot be, define notions of *weak* and *strong reliability*, and also to introduce mechanisms by which unreliability can be avoided or detected and treated as a form of error or failure. In any case, a design goal for the Ψ framework is that it should allow the exploration of other choices, both within and outside the scope of the standard semantics, and these choices are also largely orthogonal to those which must be made about how internal computation is coordinated with external interaction. Accordingly, the operational semantics is developed in a manner that preserves this freedom of choice while supporting the standard semantics. The essence of the approach is to use a process algebraic specification technique which can uniformly describe agent computation from the level of basic transitions up to an agent's top-level control cycle.

An operational semantics for a language maps terms of a syntactic domain into objects of a semantic domain, providing a complete, unambiguous specification of the meaning of program execution from which an implementation may be constructed or against which one may be validated. These semantic objects are elementary machines of some sort, commonly automata. Here, the syntactic domain will be a language of agent configurations $\mathcal{L}_\Psi$, and the objects will be, ultimately, *labelled transition systems*, explicit graphs of system behaviour, with nodes denoting system states and labelled, directed edges (transitions) denoting the possibility of particular computation steps.

Transition systems *per se* are not a very practical way of defining a computational model, especially an infinite one. Structured Operational Semantics (SOS) [15] is a well known, powerful technique which allows an infinite transition system to be specified as a finite deductive system of axioms and rules of inference by which valid transitions may be derived. However, SOS specifications are still not particularly compact, so we shall use instead *reduction systems*, a variety of conditional term rewrite system [2,12], which also permit abstraction over semantically unimportant structure of system states.

Definition 4 (*Reduction system*).

A *reduction system* $\mathcal{R}_\Upsilon$ is a calculus $\langle \Gamma, \equiv, \Xi \rangle$ consisting of three elements:

1. a *grammar* Γ of *production rules* which define a language Υ of configuration terms, which are parameterized by a signature of typed *process variables*,
2. a *structural congruence relation* $\equiv$ which partitions Υ into equivalence classes of language terms that are considered semantically identical, and
3. a set Ξ of *reduction rules* which define a *reduction relation* $\twoheadrightarrow_\Upsilon$ on configurations.

Terms of the configuration language are usually called processes. Here configurations will be compound, made up of elements of distinct types, e.g., $\langle \beta, \omega \rangle$, represented as $\beta : \omega$, and we will use process as before to refer to atomic or compound intentions ω. To distinguish them from belief transition relations we denote reduction relations by $\twoheadrightarrow$.

The function of the congruence relation, expressed as a set of equivalences between abstract terms, is to capture algebraic structure such as associativity of operators used to construct processes, and so simplify the presentation of the reduction rules, e.g., by eliminating symmetric variants. Formally, rules operate over the quotient space $\Upsilon/\!\equiv$. Reductions are thus transition relations on sets of equivalence classes of configurations, e.g., the reduction $\pi \twoheadrightarrow \omega$ stands for the relation $\{(\pi', \omega') : \pi' \equiv \pi \wedge \omega' \equiv \omega\}$.

A reduction rule is a sequent whose premises are reductions or other logical formulae and whose conclusion is a reduction. The conclusion holds, i.e., its reduction may occur, if all its premises hold, i.e., reductions in the premises are possible and logical formulae are satisfied. A rule whose premise is empty is an unconditional reduction, i.e., an axiom. A set of reduction rules inductively defines a single-step reduction relation $\twoheadrightarrow_\Upsilon$ upon configurations, namely the union of all the reductions that can be derived.

A particular feature of our approach to defining the operational semantics of Ψ is the construction of a uniform hierarchy of reduction systems, rather than the use an object language and a meta-language to capture control, cf. [6]. Each level conditionally lifts reductions at lower levels into larger contexts, and has a different notion of a single reduction step, with one step at a higher level often corresponding to many at a lower level. At the lowest level are systems which define the semantics of basic transitions and operate on configurations of belief states and intentions, and at the highest level are those which specify the control cycle and operate upon complete agent configurations.

3.1 Semantics of Intentions

It should be clear that intentions in Ψ fulfill the basic criterion for their being so named: their execution causes and determines action. Unlike intentions in BDI logics such as [18], intentions in Ψ are a commitment to a particular means rather than an end, more in the spirit of Bratman's original treatment [1]. A plan constitutes a generic, permanent commitment to behave in some way whenever any instance of a class of situations, captured by its activation trigger and guard, arises, whereas a task is a more specific, one-shot, short-term commitment to act. More abstract, goal-directed intentions may be defined directly in terms of such concrete ones [8], but we will not explore the semantics of goal-directed intention activation in this paper.

Activation is the process which replicates a plan as an active task or transforms an inactive task into an active one, capturing bindings from matching signals to triggers so that guard conditions are satisfiable. Conceptually, an enabled transition in the head of an active task is executed promptly and atomically: transition executions do not span multiple computation steps, so a transition that is reduced is fully reduced in one step, i.e., eliminated. Computations where, for example, a guard is eliminated or an action performed in a different belief state to that in which activation occurred are ruled out, as are those where multiple transitions within one intention are executed in one step. However, to achieve reliable activation and execution of parallel intentions, activation must be performed in parallel for all intentions before execution of any action occurs.

There are two reasons for this: one is that an action execution may generate new internal events, whose signalling must be deferred until the effects of current signals on all transitions has occurred[1], the other is that it may change beliefs, potentially disabling (enabling) currently enabled (disabled) transitions in other intentions. Technically, both problems may actually be avoided by use of the response state to defer actions, but there is another reason to adopt a fine-grained model which separates transition activation from execution, namely to simplify the task of selecting which enabled transitions to execute, in what order, by deferring it until the full set of them has been determined.

The key to doing this, while maintaining flexibility of semantic choice, is to separate a deterministic, choice-preserving activation step from a possibly non-deterministic, choice-making execution step. These steps are captured by separate reduction systems, whose reductions are then combined by a higher-level system which captures the overall (internal) computation policy that determines what executions occur on a particular computation step. Also, as activation must be conditional upon satisfiability of guards, it is conveniently defined in terms of the execution reductions that may occur for simple transitions. A reduction system which captures these is thus our starting point.

Intention execution. To construct a reduction system that defines intention execution for the case of simple transitions (those without a trigger), we operate on a partial agent configuration $\Psi^2 = \beta : \omega$ containing just belief and intention states. In specifying process reduction rules, we will make much use of syntactic and algebraic structure to distinguish different process types. Thus, for example, $\tau + \sigma$ can match any non-null task (as a unary task τ is congruent to $\tau + 0$) but cannot match a compound multiprocess.

[1] Ψ does not have a *micro step* semantics like that typically given to Statecharts [5]; internal events caused by transition execution do not become current until the next computation state.

Similarly, $\ddot{\sigma} \times \pi$ can match any multiprocess containing a branching task, whereas $\sigma \times \pi$ matches anything. Such expressions allow reduction rules non-deterministically to select a specific type of component from a process and apply some test or transformation.

Definition 5 (*Simple intention execution*).

The reduction system $\mathcal{R}_{\Psi^2} \subset \mathcal{R}_{\Psi}$ is defined by the grammar Γ :

$$\psi^2 ::= \beta : \omega \mid \xi : \sigma \qquad \omega ::= \pi \qquad \pi ::= \phi \mid \pi_1 \times \pi_2 \qquad \phi ::= \sigma \mid \rho \qquad \sigma ::= \mathsf{0} \mid \tau \mid \sigma_1 + \sigma_2$$

$$\rho ::= !\,\tau \qquad \tau ::= \mathcal{X} \cdot \pi \qquad \mathcal{X} ::= \mathsf{ECA} \qquad \xi ::= \mathsf{S} \qquad \mathsf{S} ::= \underline{\mathsf{E}} \qquad \mathsf{s} ::= \underline{\mathsf{e}} \qquad \delta ::= \underline{\mathsf{A}}$$

$$\mathsf{E} ::= \star \mid \dot{\mathsf{E}} \qquad \dot{\mathsf{E}} ::= \mathsf{e} \mid \mathsf{eE} \qquad \mathsf{C} ::= \star \mid \dot{\mathsf{C}} \qquad \dot{\mathsf{C}} ::= \mathsf{c} \mid \mathsf{cC} \qquad \mathsf{A} ::= \star \mid \dot{\mathsf{A}} \qquad \dot{\mathsf{A}} ::= \mathsf{a} \mid \mathsf{aA}$$

the structural congruence relation $\equiv$ defined by:

$$\pi \times \mathsf{0} \equiv \pi \qquad \pi_1 \times \pi_2 \equiv \pi_2 \times \pi_1 \qquad \pi_1 \times (\pi_2 \times \pi_3) \equiv (\pi_1 \times \pi_2) \times \pi_3$$

$$\sigma + \sigma \equiv \sigma \qquad \sigma + \mathsf{0} \equiv \sigma \qquad \sigma_1 + \sigma_2 \equiv \sigma_2 + \sigma_1 \qquad \sigma_1 + (\sigma_2 + \sigma_3) \equiv (\sigma_1 + \sigma_2) + \sigma_3$$

$$\mathsf{E} \equiv \mathsf{E}\star \qquad \mathsf{ee'E} \equiv \mathsf{e'eE} \qquad \mathsf{C} \equiv \mathsf{C}\star \qquad \mathsf{cc'C} \equiv \mathsf{c'cC} \qquad \mathsf{A} \equiv \mathsf{A}\star \qquad \mathsf{aa'A} \equiv \mathsf{a'aA}$$

and the set Ξ of reduction rules:

$$\mathsf{s}\!: \frac{\exists\theta\ (\,\mathrm{dom}(\theta) = \mathrm{var}(\mathsf{C}) \ \wedge\ \beta \xrightarrow{\mathsf{C}\theta}\,)}{\beta : \star\mathsf{CA} \cdot \pi \overset{\theta}{\twoheadrightarrow} \star : \star\star\mathsf{A}\theta \cdot \pi\theta} \qquad \mathsf{x}\!: \frac{\beta : \star\mathsf{CA} \cdot \pi \overset{\theta}{\twoheadrightarrow} \star : \tau \qquad \beta \xrightarrow{\mathsf{A}\theta} \zeta}{\beta : \star\mathsf{CA} \cdot \pi \overset{\mathsf{A}\theta}{\twoheadrightarrow} \zeta : \pi\theta}$$

$$\mathsf{c}\!: \frac{\beta : \tau \overset{\alpha}{\twoheadrightarrow} \zeta : \pi}{\beta : \tau + \sigma \overset{\alpha}{\twoheadrightarrow} \zeta : \pi} \qquad \mathsf{p}\!: \frac{\beta : \phi \overset{\alpha}{\twoheadrightarrow} \zeta : \pi}{\beta : \phi \times \omega \overset{\alpha}{\twoheadrightarrow} \zeta : \pi \times \omega} \qquad \mathsf{r}\!: \frac{\beta : \tau \overset{\alpha}{\twoheadrightarrow} \zeta : \pi}{\beta : !\tau \overset{\alpha}{\twoheadrightarrow} \zeta : !\tau \times \pi}$$

The grammar and congruences capture the set structure of compound events, conditions and actions, the algebraic properties of the combinators, and the structure of processes. Rule s checks satisfiability of the guard of a transition in the head of a unary task, and rule x combines this test with action execution. Rules c, p, and r allow this reduction to occur beneath the choice, parallel and replication operators. In the first case branches not chosen (σ) are discarded, and in the second parallel intentions (ω) persist unaffected. These rules together define a basic intention execution reduction $\beta : \dot{\pi} \overset{\alpha}{\twoheadrightarrow} \zeta : \omega$ which is non-deterministic and, except in the case where $\dot{\pi}$ is a unary task τ, unreliable.

Intention activation. A prerequisite for a consistent reliable activation semantics is that plan activation should lead to the creation of all possible tasks, while task activation should lead to just one of those possible. The rationale for this is easily seen by considering the case of a plan $!\,e(\mathrm{x})\star\star \cdot \pi$. Suppose the events $e(1)$ and $e(2)$, of the same type but with different properties, are signalled on successive cycles. Both will cause plan activation, creating tasks with x instantiated to 1 and 2 respectively. So if by chance the two events are signalled on the same cycle, then both activations should occur. By contrast, if activation is reliable, the task $e(\mathrm{x})\star\star \cdot \pi$ will be activated by the first signal to occur but not by the second, since by then it will already have reduced to $\pi\theta$, hence if they are signalled simultaneously, either but not both activations should occur. Another way to think of this is that a plan with free variables in its activation trigger is equivalent to a parallel composition of all distinct instantiations of these, whereas a corresponding task is equivalent to a choice between all such instantiations.

The key to capturing this semantics formally is to model task activation as a reduction $\sigma \twoheadrightarrow \sigma + \sigma'$ of a task to one whose additional branches represent its possible activations, any one of which may be chosen, and plan activation as a reduction $\rho \twoheadrightarrow \rho \times \pi$ of a plan to one in parallel with its possible activations, all of which may be executed.

Definition 6 (*Intention activation*).

The reduction system $\mathcal{R}_{\Psi^3}$ extends $\mathcal{R}_{\Psi^2}$ to include the signal state ξ and the rules:

$$\mathsf{st}: \frac{\mathcal{X} = \mathsf{eECA} \quad \exists\theta\,(\,\mathrm{dom}(\theta) = \mathrm{var}(\mathsf{e}) \;\wedge\; \mathsf{s} = \mathsf{e}\theta\,)}{\mathsf{s} : \mathcal{X} \cdot \pi \twoheadrightarrow \star : \mathcal{X} \cdot \pi \;+\; (\mathsf{ECA} \cdot \pi)\theta} \qquad \mathsf{ut}: \frac{\mathsf{s} : \tau \not\twoheadrightarrow}{\mathsf{s} : \tau \twoheadrightarrow \star : \tau}$$

$$\mathsf{b}: \frac{\mathsf{s} : \tau \twoheadrightarrow \star : \sigma_1 \quad \mathsf{s} : \sigma \twoheadrightarrow \star : \sigma_2}{\mathsf{s} : (\tau + \sigma) \twoheadrightarrow \star : (\sigma_1 + \sigma_2)} \qquad \mathsf{m}: \frac{\mathsf{s} : \sigma \twoheadrightarrow \star : \sigma_1 \quad \mathsf{S} : \sigma_1 \twoheadrightarrow^{=} \star : (\sigma + \sigma_2)}{\mathsf{sS} : \sigma \twoheadrightarrow \star : (\sigma + \sigma_2)}$$

$$\psi^3 ::= \xi : \psi^2 \qquad \mathsf{ta}: \frac{\xi : \sigma \twoheadrightarrow \sigma + \sigma_1 + \sum_{i=1}^{n} \tau_i \qquad \beta : \sigma_1 \not\twoheadrightarrow \qquad \forall \tau_i\, \beta : \tau_i \twoheadrightarrow}{\xi : \beta : \sigma \twoheadrightarrow \star : \beta : \sigma + \sum_{i=1}^{n} \tau_i}$$

$$\mathsf{pa}: \frac{\xi : \beta : \tau \twoheadrightarrow \star : \beta : \tau + \sum_{i=1}^{n} \tau_i}{\xi : \beta : !\tau \twoheadrightarrow \star : \beta : !\tau \times \prod_{i=1}^{n} \tau_i} \qquad \mathsf{a}: \frac{\xi : \beta : \phi \twoheadrightarrow \star : \beta : \pi_1 \quad \xi : \beta : \pi \twoheadrightarrow \star : \beta : \pi_2}{\xi : \beta : \phi \times \pi \twoheadrightarrow \xi : \beta : \pi_1 \times \pi_2}$$

Rules st and ut define successful and unsuccessful *triggering* of a unary task $\tau = \mathcal{X} \cdot \pi$ by a single signal s; if successful, the task reduces to a choice between itself and a derivative to which the matching substitution has been applied and whose head has been (perhaps only partially) simplified, if not it remains unchanged, and in both cases the signal is consumed. Rule b extends triggering to branching tasks by recursively performing it in parallel on all branches. Unlike typical reductions under choice, all alternatives are retained. Rule m defines multiple triggering, recursively applying all signals in a single step. The branches of the resulting task are, at minimum, just those of the original ($\sigma_2 = \mathsf{0}$) when none of the signals can match any events in the trigger; otherwise, a choice between the original task and all of its (possibly partially) simplified derivatives. For example, letting e and o be distinct event types:

$$e(3)\,e(2)\,o() : e(\mathrm{x})\,o()\star a(\mathrm{x}) \;\twoheadrightarrow_{\mathsf{m}}$$
$$\star : e(\mathrm{x})o()\star a(\mathrm{x}) \;+\; e(\mathrm{x})\star a(\mathrm{x}) \;+\; o()\star a(2) \;+\; \star\star a(2) \;+\; o()\star a(3) \;+\; \star\star a(3)$$

Here an unguarded unary task (in fact, a rule $\mathcal{X} \cdot \mathsf{0}$ written as $\mathcal{X}$) with a binary trigger is triggered by a compound signal $e(3)\,e(2)\,o()$ which can match it in two distinct ways. Triggering as defined by these four rules generates a branching task that preserves all possible choices arising from non-determinism due to multiple signals, including the choice "none of the above" represented by the original task. However, it may generate incompletely simplified branches which are thus not enabled and must be discarded. Rule ta defines task activation, lifting triggering into Ψ^3 and filtering out such branches, and eliminating those not enabled in the current belief state (not reducible in Ψ^2). Extending the example to add a guard condition, and assuming $[\![\mathrm{u}]\!] = 2$, we obtain:

$$e(3)\,e(2)\,o() : \beta : e(\mathrm{x})\,o()[\mathrm{x} > \mathrm{u}]a(\mathrm{x}) \;\twoheadrightarrow_{\mathsf{ta}}\; \star : \beta : e(\mathrm{x})\,o()[\mathrm{x} > \mathrm{u}]a(\mathrm{x}) \;+\; \star[3 > \mathrm{u}]a(3)$$

i.e., a choice between the original task and its only satisfiable activation. The effect of this rule is to transform a task in one step into one which is a choice between the original and all possible enabled, complete simplifications of any branch of the original. It does not, however, choose between any distinct substitutions that might satisfy guards; any such choice is made subsequently during execution.

Rule pa defines plan activation, i.e., the behaviour of activation beneath replication: if a unary task τ is activated to itself or a (possibly empty) sum, then the corresponding plan $!\,\tau$ is activated to itself in parallel with a (possibly empty) product. The behaviour of transitions with compound triggers should be noted. Corresponding simplified branches are generated only if all the events in the compound trigger are matched, so detecting simultaneity of signals. If there is more than one way that a set of signals can match the trigger, then a fully simplified branch or task is generated for each distinct matching subset. Finally, rule a extends activation to compound multiprocesses by activating all intentions in parallel.

These rules are a remarkable example of the expressive power of reduction systems and show their ability to capture computational processes in a compact and lucid way. The reduction defined by $\mathcal{R}_{\Psi^3}$ is of the general form $\dot{\xi} : \beta : \dot{\pi} \twoheadrightarrow \star : \beta : \omega$, and for any non-empty signal state and non-null intention is unconditional and deterministic, merely consuming the signal state when no matches occur (a vacuous activation). It is monotonic w.r.t. to the signal state, and compositional w.r.t. the operators $+$ and $\times$. By performing a possibly vacuous, unconditional, deterministic reduction on the heads of all branches of all intentions in a single, parallel step, it provably captures the broadcast and *must* semantics required by reliability [8,9]. Thus it constitutes a fundamental semantic building block, however, it may when desired be selectively applied so as to weaken these semantic guarantees.

As mentioned above, ensuring reliable execution is impossible in the general case. For example, the process $\mathrm{u} := 1 \cdot 0 \times \mathrm{u} := 2 \cdot 0$ can never be reliably executed; it constitutes a simplest *inconsistent intention state*, where an inherent conflict between actions prevents their co-execution. We consider such a situation to be a specification error. By contrast, the process $[\mathrm{u} > 0]\{\mathrm{u} := 0, \mathrm{u}_1 := 1\} \cdot 0 \times [\mathrm{u} > 0]\{\mathrm{u} := 0, \mathrm{u}_2 := 2\} \cdot 0$ can be executed reliably if both intentions are executed in one parallel step, but not by any *minimal execution policy* such as uncontrolled interleaving which, no matter how it selects among enabled intentions, executes at most one intention per step.

The problem here is not any inconsistency of the intention state, but a lack of serializability which arises because whichever intention is executed first immediately disables the other. Yet disabling the condition which activates such condition-action rules is crucial for their correct behaviour, to avoid their being repeatedly activated in successive states. As we shall see below, reliable execution can be achieved whenever there is no unavoidable conflict between actions by adopting a step semantics for intention execution that avoids avoidable conflicts by more intelligently resolving sources of non-determinism.

3.2 Agent Computation

Having defined the semantics of intention activation and simple intention execution, we can now assemble a framework for agent computation based upon these reductions, extending the configuration to include the response state and the event and action queues.

Definition 7 (*Agent computation system $\mathcal{R}_\Psi$*).

The agent computation system $\mathcal{R}_\Psi$ extends $\mathcal{R}_{\Psi^3}$ with the following rules:

$$\psi ::= \psi^6 \mid \psi^4 \mid \psi^3 \mid \psi^2 \qquad \psi^6 ::= \mathcal{Q}_{\mathsf{E}} : \psi^4 : \mathcal{Q}_{\mathsf{A}} \qquad \mathcal{Q}_{\mathsf{E}} ::= \underline{\mathsf{E_e}}^* \qquad \mathcal{Q}_{\mathsf{A}} ::= \underline{\mathsf{A_e}}^*$$

$$\psi^4 ::= \psi^3 : \delta \qquad \mathsf{min}: \frac{\xi : \beta : \pi \twoheadrightarrow^{=} \star : \beta : \pi_1 \quad \beta : \pi_1 \xrightarrow{\mathsf{A}}\!\!\!\twoheadrightarrow \zeta : \omega \quad \beta \xrightarrow{\delta \cup \mathsf{A}}}{\xi : \beta : \pi : \delta \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \omega : \delta \cup \mathsf{A}}$$

$$\mathsf{int}: \frac{\xi : \beta : \pi : \star \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \omega : \delta \quad \beta \xrightarrow{\delta} \zeta \quad \mathsf{E} = \delta \cap \mathcal{L}_{\mathsf{E_i}} \quad \mathsf{A} = \delta \cap \mathcal{L}_{\mathsf{A_e}}}{\mathcal{Q}_{\mathsf{E}} : \xi : \beta : \pi : \delta_0 : \mathcal{Q}_{\mathsf{A}} \xrightarrow{\mathsf{A}}\!\!\!\twoheadrightarrow \mathcal{Q}_{\mathsf{E}} : \mathsf{E} : \zeta : \omega : \delta : \mathcal{Q}_{\mathsf{A}} \otimes \langle \mathsf{A} \rangle}$$

$$\mathsf{obs}: \frac{\beta \xrightarrow{\mathsf{E}}\!\!\!\twoheadrightarrow \zeta}{\langle \mathsf{E} \rangle \otimes \mathcal{Q}_{\mathsf{E}} : \xi : \beta : \omega : \delta : \mathcal{Q}_{\mathsf{A}} \longrightarrow\!\!\!\!\twoheadrightarrow \mathcal{Q}_{\mathsf{E}} : \xi \cup \mathsf{E} : \zeta : \omega : \delta : \mathcal{Q}_{\mathsf{A}}}$$

This system operates at two configuration levels simultaneously. First, a reduction relation on Ψ^4 of the general form $\xi : \beta : \pi : \star \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \omega : \delta$ selects which intentions to activate and execute, accumulating and deferring any actions into the response state δ. This constitutes an *internal computation policy*. Second, a reduction relation defined on the full configuration Ψ^6, which constitutes an *embedded computation policy*, coordinates internal computation with observation, belief update and external action.

In this particular example, the single rule min defines the policy for internal computation, merely composing activation of all intentions with execution of just one. Rule int defines an entire computation step, lifting reductions of Ψ^4, performing belief update on the basis of the response state δ, placing any external actions on the action queue and extracting and directly signalling any new internal events. Rule obs performs *observation*, taking a group of perceived events from the event queue, performing their epistemic effects and signalling them. Here, there are no constraints on when observation occurs and some *observation policy* is likely required; two possibilities are that it should only occur when the signal state is empty, or when there are also no enabled intentions; such constraints may easily be imposed by adding suitable premises to rule obs.

The system embodies specific choices about how to resolve non-determinism, but quite different choices may easily be made. For example, by replacing min with the rules below, one obtains a step semantics for intentions which executes in parallel a maximal, conflict free subset of the enabled intentions, and so achieves *weak reliability*.

Definition 8 (*Simple maximal process execution*).

$$\mathsf{mx1}: \frac{\beta : \phi \xrightarrow{\mathsf{A}}\!\!\!\twoheadrightarrow \zeta : \pi \quad \beta \xrightarrow{\delta \cup \mathsf{A}}}{\star : \beta : \phi : \delta \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \pi : \delta \cup \mathsf{A}}$$

$$\mathsf{mx2}: \frac{\star : \beta : \phi : \delta \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \pi_1 : \delta_1 \quad \star : \beta : \pi : \delta_1 \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \pi_2 : \delta_2}{\star : \beta : \phi \times \pi : \delta \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \pi_1 \times \pi_2 : \delta_2}$$

$$\mathsf{mx3}: \frac{\star : \beta : \phi : \delta \not\twoheadrightarrow \quad \star : \beta : \pi : \delta \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \pi_2 : \delta_2}{\star : \beta : \phi \times \pi : \delta \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \phi \times \pi_2 : \delta_2}$$

$$\mathsf{max}: \frac{\xi : \beta : \pi \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \pi_1 \quad \star : \beta : \pi_1 : \star \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \pi_2 : \delta}{\xi : \beta : \pi : \star \longrightarrow\!\!\!\!\twoheadrightarrow \star : \beta : \pi_2 : \delta}$$

The three mx rules recursively define a maximal execution reduction. The first handles the base case; the second the recursive case where an intention is executable, in which case it is reduced, and the third that where it is not, in which case it remains unchanged. The latter case may occur for any of three reasons: the intention is not enabled, it is enabled but not executable, or its action cannot be co-executed with the response set already chosen. The reduction occurs provided at least one intention is executable, and is non-deterministic in the general case since co-executability may depend on the (random) order in which intentions are considered. Rule max merely composes this reduction with an activation reduction.

Every reduction permitted by this system is maximal in that it cannot be extended with further intention executions. However, it is not strongly reliable because it makes arbitrary choices among the co-executable branches of tasks without regard to their consequences on other intentions. To do better and achieve stronger forms of reliability is feasible, but requires more complex reduction systems, as does the definition of a sensible scheme for detecting and recovering from various forms of intention failure, such as local deadlock. The details of such control policies, and formal definitions of degrees of reliability and intention consistency may be found elsewhere [8].

The constraints that the system $\mathcal{R}_\Psi$ imposes are in fact rather minimal. For example, it is straightforward, if desired, to implement control policies which selectively broadcast signals to particular intentions rather than to all, or which serially execute enabled intentions in some particular order over multiple computation steps. The framework thus achieves the goal of permitting a flexible choice of agent computation semantics, while simultaneously enabling the standard semantics of reliable computation.

4 Conclusions and Acknowledgements

We have presented the core of a novel algebraic agent language – the Ψ calculus – which captures the full complexity of agent architectures that employ a sense–compute–act computation cycle and are based on stored plan execution, and demonstrated how its operational semantics may be exactly defined. Viewed abstractly, Ψ is a broadcast calculus with shared mutable state and buffered asynchronous interfaces for perception of and action upon an external environment. Its semantics is specified, not in a typical logic setting, but in a uniform algebraic style by reduction systems, a variety of conditional term rewrite system [2,12]. We have tried here to distinguish clearly between the basic agent model and semantic structure of the Ψ framework, and the semantics which results from adopting a specific set of policies for the control of computation, identifying one such set as capturing a standard semantics preferred on the basis of its guarantee properties, and the intuitive, synchronous computation model which results. Others' taste in such matters may differ, so the framework is designed to allow a range of control policies to be easily constructed.

Ψ is a complex calculus, and the presentation here is partial. Some of the complexity arises from the generality and flexibility of the framework, and some because we have deliberately avoided the use of powerful multi-step reductions to keep the exposition of its semantics explicit and less demanding of a deep understanding of reduction systems; however, the complexity is due in no small part to the intrinsic complexity of plan execution systems. The Ψ plan language $\mathcal{L}_P$ and its standard semantics, in contrast to the systems which specify that semantics, are uniform, intuitive and simple: plans are built up from transitions, denoting triggerable, guarded actions which execute atomically, by means of sequencing and choice in most common cases, with forking and embedded replication available as advanced programming techniques. Unlike its predecessors PRS and dMARS, there are no special cases: any transition may be a triggered, guarded compound action, and a plan's activation transition is just one such case.

By defining an intention language which is both general and regular, with very few restrictions on how processes are composed, several benefits ensue. The task of defining or understanding a particular semantics is arguably simplified, and many restricted languages, e.g., those based on rules or limited to singleton triggers in transitions, are treated naturally as special cases. The algebraic structure of the language is exposed for study, and comparison with other languages and formalisms is simplified. One is also forced, as a designer, to develop a semantic treatment of the general case which respects the uniformity of the language, and in doing so to acquire a more profound understanding of the intricacies and possibilities of plan execution systems.

In any system such as Ψ, where multiple processes operate upon shared mutable state, one cannot avoid non-compositionality of processes in the general case except by imposing onerous restrictions on process structure. Indeed, non-compositionality is essential if sets of coordinated, interacting processes are to be constructed. However, one can if one chooses avoid it in many cases where it arises needlessly between non-interacting processes, by adopting suitably designed policies for the control of computation. In essence, one constructs a computational model which provides the strongest possible guarantees of predictability of execution that can tractably be obtained. This can be done by exploiting and resolving available non-determinism in a manner that avoids computations where intended properties, such as the postconditions of actions, are violated. Of course, this requires reasoning in some manner about the pre- and postconditions of actions, which must done without excessive runtime overhead.

The standard semantics of Ψ follows this approach and is strongly fair, promising that *everything that should happen will happen, promptly and in parallel*, rather than *things that could happen might happen, eventually, in some unpredictable order*. Cases where this cannot be achieved are treated as failures, except where the programmer has explicitly given her permission for this guarantee to be relaxed [8]. This computation model means that an agent programmer need no longer worry about the order in which intention steps are interleaved, or be forced to group all the activity that should occur in response to a particular event into a single plan. Examining a set of plans, she can reason about their behaviour in terms of a 1-to-1 correspondence between the local states of enabled intentions and global system states. We would argue that the language, while more powerful and general than its predecessors, is thus significantly easier to understand and reason about. Further exploration of the formal properties of the standard semantics, and those which result from other control policies, is an area where more research is highly desirable. We hope that others will take up the challenge of exploring these issues, and will find the Ψ calculus a useful contribution to that activity.

The author thanks Frank Dignum, Bob Kowalski and Chris Wright for their helpful feedback and suggestions on previous drafts, and especially thanks Liz Sonenberg and Noriko Hayashi for their unfailing encouragement and support for this work.

References

1. Michael E. Bratman. *Intentions, Plans, and Practical Reason.* Harvard University Press, 1987.
2. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science Vol. B*, pages 243–320. MIT Press/Elsevier, 1990.
3. M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In *Intelligent Agents IV: Proceedings of ATAL-97*, Providence, RI, 1997. Springer LNAI 1365.
4. Michael P. Georgeff and Felix Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of IJCAI-89*, pages 972–978, Detroit, MI, 1989.
5. D. Harel and C. Kahana. On statecharts with overlapping. *ACM Transactions on Software Engineering and Methodology*, 1(4), 1992.
6. Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Control structures of rule-based agent languages. In *Intelligent Agents V: Proceedings of ATAL-98*, pages 381–396, Paris, 1998. Springer LNAI 1555.
7. David Kinny. *The Distributed Multi-Agent Reasoning System Architecture and Language Specification*. Australian Artificial Intelligence Institute, Melbourne, Australia, 1993.
8. David Kinny. *Fundamentals of Agent Computation Theory: Semantics*. PhD thesis, Department of Computer Science, University of Melbourne, Australia, 2001.
9. David Kinny. Reliable agent computation: an algebraic perspective. In *Proceedings of the 4th Pacific Rim Workshop on MultiAgent Systems*, Taipei, Taiwan, 2001. Springer.
10. Yves Lesperance, Hector J. Levesque, Fangzhen Lin, Daniel Marcu, Raymond Reiter, and Richard B. Scherl. Foundations of a logical approach to agent programming. In *Intelligent Agents II: Proceedings of ATAL-95*, pages 331–346, Montréal, 1995. Springer LNAI 1037.
11. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1992.
12. José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
13. Robin Milner. The polyadic π-calculus: A tutorial. Technical Report LFCS report 91-180, University of Edinburgh, Edinburgh, 1991.
14. Jörg P. Müller. The right agent (architecture) to do the right thing. In *Intelligent Agents V: Proceedings of ATAL-98*, pages 211–225, Paris, 1998. Springer LNAI 1555.
15. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.
16. K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25, 1995.
17. Anand Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Agents Breaking Away: Proceedings of MAAMAW '96*, Eindhoven, The Netherlands, 1996. Springer LNAI 1038.
18. Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning, KR '91*, pages 473–484, Cambridge, MA, 1991.
19. Yoav Shoham. AGENT0: a simple agent language and its interpreter. In *Proceedings of the Ninth National Conference on Artificial Intelligence, AAAI-91*, pages 704–709, Los Angeles, CA, 1991.
20. Gerd Wagner. A logical and operational model of scalable knowledge- and perception-based agents. In *Agents Breaking Away: Proceedings of MAAMAW '96*, pages 26–41, Eindhoven, The Netherlands, 1996. Springer LNAI 1038.
21. Jennifer Widom and Stefano Ceri. *Active Database Systems*. Morgan Kaufmann, 1996.

Evolving Real-Time Local Agent Control for Large-Scale Multi-agent Systems*

Thomas Wagner[1] and Victor Lesser[2]

[1] Automated Reasoning Group
Honeywell Laboratories
3660 Technology Drive, MN65-2600
Minneapolis, MN 55418
wagner_tom@htc.honeywell.com
http://www.drtomwagner.com

[2] Computer Science Department
University of Massachusetts
Amherst, MA 01003
lesser@cs.umass.edu
http://mas.cs.umass.edu

Abstract. Control for agents situated in multi-agent systems is a complex problem. This is particularly true in hard, open, dynamic environments where resource, privacy, bandwidth, and computational limitations impose restrictions on the type of information that agents may share and the control problem solving options available to agents. The *MQ* or *motivational quantities* framework addresses these issues by evaluating candidate tasks based on the agent's organizational context and by framing control as a local agent optimization problem that approximates the global problem through the use of state and preference.

1 Introduction

Many researchers believe that one of the dominant future models of distributed computation will involve large networks of interacting heterogenous software agents. We, as a community, are showing significant progress in making this a reality but many research questions remain. Consider the requirements and characteristics of the problem space. The overall objective is to create *open*, *large-*

* Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreement number #F30602-99-2-0525; effort also supported by the National Science Foundation under Grant No. IIS-9812755. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory, National Science Foundation, or the U.S. Government.

scale, information and computational systems that are flexible, adaptable, robust, persistent, and autonomous. Now, consider the implications of this. Openness means that agents may interact freely, come and go from the network, and that the entire problem solving environment is dynamic. Openness thus often acts to thwart agent technologies that rely on detailed predictability or static properties of the problem space. In our work, we take the view that openness leads to a requirement for real-time agent control problem solving so that agents can respond to change and unexpected outcomes online.

Moving the scale of multi-agent systems from small groups to large groups, e.g., tens of thousands, throws two other problems into the mix: *increased interaction overhead* and *social complexity.* The term interaction overhead denotes the increase in communication between agents required to detect interactions in their problem solving and to coordinate their activities, i.e., it denotes the sheer volume of message traffic and problem solving required to evaluate the messages. This is being dealt with by imposing organizational structure on the agents so that they do not all communicate and by creating coordination and communication technologies that are adjustable [6,10,38,11]. The other issue is social complexity and we do not mean social complexity in the human sense. Or rather, the goal of this research is not to study social complexity in human organizations [33] per se as our work in agent control has very specific task-centered properties. When agents are situated in a large open environment, and organizational structure is imposed upon them, they have different organizational objectives and they must reason about how their problem solving relates to satisfying their multiple, and possibly conflicting, organizational objectives.

This research focuses on exactly this problem – how agents in large-scale open environments reason about their organizational context and make appropriate choices about which actions to perform and how to go about performing them. It is important to emphasize that this research pertains to complex problem solving agents, e.g, the BIG Information Gathering Agent [28] and [23,6,19], where the agents are situated in an environment, able to sense and effect, and have explicit representations of candidate tasks and explicit representations of different ways to go about performing the tasks. Additionally, tasks are quantified or have different performance characteristics and, following in the thread of complex problem solving [14,9,31] there are relationships between the tasks. The implications are that tasks cannot be reasoned about independently and that the value or utility of particular tasks differs depending on the context. We call the process of reasoning about which tasks to perform, when, with what resources, how or in what fashion, and with whom to coordinate, the *local agent control* problem. The term "local" is used in this expression because agency, as we use it, denotes an autonomous distributed problem solving entity. In our work, there is no global picture[1] of all activities being carried out by all agents nor are the

[1] In multi-agent systems we take the position that it is not generally possible to compose a global picture of the activities happening at all the agents. This is due to the combinatorics involved in gathering the information and reasoning about it. It is also due to privacy, intellectual property, and autonomy issues. For example, an

agents situated in specialized, tightly coupled environments like Tambe's teams [38] or robotic soccer [40].

We view local agent control in this context as a real-time action-selection-sequencing problem where an agent has n candidate tasks and alternative different ways to perform the tasks. Tasks have deadlines and other constraints as well as different performance properties, e.g., consuming different resources or producing results of varying quality. Control in this context is an optimization problem where different solutions are possible and they have different degrees of utility.

Historically in our work this class of control problem has been dealt with using the TÆMS task modeling framework [7,27], GPGP coordination [7], and Design-to-Criteria (DTC) real-time agent scheduling [34,43,42,46,41]. Using these tools, an individual agent for use in a multi-agent environment is constructed by coupling a domain expert or planner with GPGP and DTC. In this model, the domain expert's function is to perform domain problem solving and to translate its internal representations into TÆMS for control problem solving by the coordination (GPGP) and trade-off/scheduling (DTC) experts. GPGP and DTC then work together to guide the actions of the individual agent and to coordinate the activities of the agent with the other agents in the network. This is the approach used in the BIG information gathering agent [29,28], the Intelligent Home project (IHome) [26], the DARPA ANTS real-time agent sensor network for vehicle tracking [41,19], and others [48]. Though in some of these applications GPGP is replaced by other communication machinery that forms commitments between agents about which tasks will be performed and when. In all of these applications, DTC or its predecessor, Design-to-Time [16], is the oracle that guides and constrains the communication and commitment formation processes.

TÆMS and DTC are mature research artifacts and have been successfully reused in many applications (DTC since 1995). However, TÆMS is not suited to addressing the situational complexity that arises when agents are deployed in larger groups or in open environments. One of the fundamental limitations of TÆMS is that it is a static representation of an agent's problem solving process at a given instant in time. It is, in essence, a snapshot of the options available to the agent and a snapshot of their characteristics. In our applications, generally, when the situation changes and the characteristics of tasks (used to determine utility) change, the problem solver must adjust the performance profiles and emit a new TÆMS task structure. Another limitation is that in TÆMS, action performance produces *quality* which then propagates throughout the entire graph-like structure in ways that is intended to model distributed problem solving as in a distributed interpretation problem [13]. The formal details of TÆMS are in [8]. While this view is appropriate for reasoning about interrelated domain problem solving activities at a detailed level, it is not readily used to model concepts like tasks that contribute to one organizational objective while being detrimental to another. TÆMS also does not adequately support concepts like the value of

agent affiliated with Microsoft is unlikely to share its entire knowledge base with a Department of Justice agent even if it were computationally feasible.

forming a commitment with another agent or the penalty for decommitting from an activity once a commitment is formed.

To address these limitations, we have developed a new framework for representing tasks and actions at a different level of abstraction. The framework, called the *motivational quantities* (MQ) [39,45,44,47] framework, uses state to achieve "automatic" changes in task valuation or utility (unlike the static view taken in TÆMS). The MQ framework also describes tasks in many different attribute dimensions so that we can model tasks contributing to, or detracting from, different objectives to different degrees. While control at the TÆMS level pertains to detailed evaluation of domain problem solving activities of an agent, control at the MQ level pertains to high-level valuation of candidate tasks based on an understanding of the relationship between tasks and organizational objectives. In other words, in the MQ framework, task value is determined not only by the intrinsic properties of tasks, but by the benefits and costs of the intrinsic properties as determined by the agent's current organizational situation. From another view, there is an intermediate evaluation step in the control process whereas such processes typically focus on intrinsic value rather than contextually interpreted value. While we have ideas about how to combine and interface [39] the two levels, integration is clearly unnecessary for many applications.

A preliminary version of the MQ framework was presented in [45]. In this paper we refine the framework based on experiences gained by implementing and working with the model.

2 Quantifying and Comparing Motivations

In the MQ model we make the control restriction that for an agent to perform a task, or to consider a task, the task must produce value for the local agent. On the surface, this implies that the MQ model is only for controlling interacting self-interested agents. This is not the case. The restriction is to guarantee the ability to compare tasks from a unified perspective. Consider the issue of task value. When agents are isolated problem solving entities, task performance produces value that is entirely of local benefit. In multi-agent systems, value may be of local benefit and of benefit to other agents. The extremes are also possible; tasks may be only of local benefit and tasks may be only of benefit to agents other than the local agent. This latter case appears problematic for the control restriction above: *all tasks produce local value.* This case is problematic only on the surface. For the local agent to consider performing such a task, it must indeed have value, however, in this case the value is of a different type or class than the value of its other candidate tasks. The task, for example, may be performed to meet some organizational directive, e.g., *service requests from agent* β, or to reduce favors owed to the agent, to accumulate favors for future use with the agent, or because a different agent with which the local agent holds common goals requested it.

In the MQ framework, all tasks have value or a *motivation* for performing the task where the value is determined both by the value of the task and by the importance of the organizational objective with which the task is associated (and the current state of goal achievement). This enables the agent to compare and value tasks that are associated with different organizational goals, or tasks that are detrimental to one organizational goal while having positive benefit to a different organizational goal, or tasks associated with different organizations entirely, or tasks motivated by self-interested reasons to cooperative reasons. The MQ framework quantifies these different underlying motivational factors and provides the means to compare them via a multi-attributed utility function. Definitions:

Agents are autonomous, heterogenous, persistent, computing entities that have the ability to choose which tasks to perform and when to perform them. Agents are also rationally bounded, resource bounded, and have limited knowledge of other agents.[2] Agents can perform tasks locally if they have sufficient resources and they may interact with other agents. Additionally:

– Each agent has a set of MQs or *motivational quantities* that it tracks and accumulates. MQs represent progress toward organizational goals[3]. MQs are produced and consumed by task performance where the consumption or production properties are dependent on the context. For example, two agents interacting to achieve a shared organizational goal may both see an increase in the same local MQ levels as progress is made (this is not a zero sum game), whereas agents interacting to satisfy different goals may each obtain different types and quantities of MQs from the same interaction.
– Not all agents have the same MQ set. However, for two agents to form a commitment to a specific course of action, they must have at least one MQ in common (or have the means for forming an MQ dynamically). If they do not have an MQ in common, they lack any common goals or objectives and lack any common medium of exchange. (Proxy and reducibility are somewhat addressed in [45].)
– For each MQ_i belonging to an agent, it has a preference function or utility curve, U_{f_i}, that describes its preference for a particular quantity of the MQ, i.e., $\forall MQ_i,\ \exists U_{f_i}()$ such that $U_{f_i}(MQ_i) \mapsto U_i$ where U_i is the utility associated with MQ_i and is not directly interchangeable with U_j unless $i = j$. Different agents may have different preferences for the same MQ_i. Preferences in the framework are defined by the relation between task performance and organizational goals or directives.
– An agent's overall utility at any given moment in time is a function of its different utilities: $U_{agent} = \gamma(U_i,\ U_j,\ U_k,\ ..)$. We make no assumptions about the properties of $\gamma()$, only that it enables agents to determine preference or dominance between two different agent states with respect to MQs.
– For simplicity of presentation, let us assume that $\gamma()$ is not a multi-variate utility function and instead that for each U_i there is an associated function

[2] As agents are heterogenous, they may be associated with different corporate entities (privacy issues), and because the contextual valuation of tasks is generally an exponential problem we do not assume agents know each other's utility functions, plan libraries, etc.

[3] In certain cases, MQs may also be used as a medium of exchange. Though little meaningful work has been done to explore this.

$\omega_i()$ [4] that translates MQ specific utility into the agent's general utility type, i.e., $\forall U_i,\ \exists \omega_i()$ such that $\omega_i(U_i) \mapsto U_{agent}$. Thus U_{agent} may take the form of $U_{agent} = \sum_{i=0}^{n} \omega_i(U_i)$.
– Change in agent utility, denoted ΔU_{agent}, is computed through changes to the individual utilities, U_i, U_j, etc. Let U_i denote the utility associated with MQ_i before the quantity of the MQ changes (e.g., as the result of task performance). Let U_i' denote the utility associated with the changed MQ quantity. The change in overall utility to the agent, in this simplified model, is expressed as $\Delta U_{agent} = |\sum_{i=0}^{n} \omega_i(U_i') - \omega_i(U_i)|$

MQ Tasks are abstractions of the primitive actions that the agent may carry out. MQ tasks:

– May have deadlines, $deadline_i$, for task performance beyond which performance of said task yields no useful results. (This could also be defined via a function that describes a gradual decrease in utility as $deadline_i$ passes.) This temporal constraint, as with the one following, is particularly important for multi-agent coordination and temporal sequencing of activities over interactions.
– May have earliest start times, $start_i$, for task performance before which performance of said task yields no useful results. (This could also be defined via a function that describes a gradual increase in utility as $start_i$ approaches.)
– Each MQ task consists of one or more MQ alternatives, where one alternative corresponds to a different performance profile of the task. In many ways, this extension simplifies reasoning with the preliminary model presented in [45] while at the same time increasing the representational power of the framework by coupling different durations with the other performance characteristics. Each alternative:
  • Requires some time or duration to execute, denoted d_i. The durations for all the alternatives of the task may be the same as the different alternatives may differ in other ways (below). Deadlines and start time constraints remain at the task level – the idea being that tasks have constraints that apply to all of the alternatives.
  • Produces some quantity of one or more MQs, called an *MQ production set* ($MQPS$), which is denoted by: $MQPS_{i,j,k} = \{q_i, q_j, q_k, ..\}$, where $\forall i,\ q_i \geq 0$. These quantities are *positive* and reflect the benefit derived from performing the task, e.g., progress toward a goal or the production of an artifact that can be exchanged with other agents. In this model, the two are equivalent.
  • Akin to the $MQPS$, tasks may also consume quantities of MQs. The specification of the MQs consumed by a task is called an *MQ consumption set* and denoted $MQCS_{i,j,k} = \{q_i, q_j, q_k, ..\}$, where $\forall i,\ q_i \leq 0$. Consumption sets model tasks consuming resources, or being detrimental to an organizational objective, or agents contracting work out to

[4] $\omega_i()$ could be combined with $U_{f_i}()$. These are partitioned for mapping different organizational influences.

other agents, e.g., paying another agent to produce some desired result or another agent accumulating favors or good will as the result of task performance. Consumption sets are the *negative* side of task performance.
    - All quantities, e.g., d_i, $MQPS$, $MQCS$, are currently viewed from an expected value standpoint.
- Note that the $MQPS_{alternative\_i} \cap MQPS_{alternative\_j}$ may $\neq \phi$ as different alternatives may have common members. This is also true for the $MQCS$. The reason for this is that alternatives may represent producing different degrees of benefit (MQ levels) toward an objective as well as simply producing benefits toward different objectives (different MQs).
- Tasks whose performance at a given point in time, t, will miss their deadlines should not be performed. Likewise with tasks whose performance violates their start time constraint. If such tasks are executed: 1) all $MQCS$ will apply, 2) no $MQPS$ will apply, 3) tasks will take their full duration to execute. Conceptually, this models performing the task, and consuming the task's resources, but having the task fail to produce any benefit.
- In any given alternative, $MQPS$ and $MQCS$ must be disjoint. The reason for this restriction is that in order to reason about an alternative producing and consuming the same MQs, we must have a detailed model of the execution characteristics of the alternative. For example, we must know when it consumes (at the beginning, at the end, linearly across the execution, etc.) and when it produces. This is not consistent with the MQ task abstraction – for situations in which such detailed reasoning is desired, the MQ task must be broken into multiple different tasks.
- $MQCS$ defines quantities that are required for task performance. If a task lacks sufficient MQs for execution it is deemed un-executable and will not be performed in any fashion. This means it will have zero duration, consume zero MQs, and will produce zero MQs.
- If a task will both violate a deadline/start time constraint and lacks sufficient resources to execute, the $MQCS$-lacking semantics will apply. The rationale is that the task lacks the resources to begin execution and thus does not actually violate the temporal constraints.
- Tasks may be interrupted, however, when this occurs they consume all MQs in the $MQCS$ and produce none of the MQs in the $MQPS$. This restriction is to simplify the semantics for the reasoning process.

In this section we have presented a model for comparing tasks that are motivated by different factors. The model can support comparison between tasks that are performed for different organizational motivations to task that are performed for other agents in return for financial gain to tasks that are performed for other agents for cooperative reasons. Via the different preferences for the different quantities, agent control can be modulated and agents can reason about mixtures of different task types and different motivations. The use of state in the model also facilitates contextually dependent behaviors or adjustments to behaviors over time. Agent α performing cooperative work with a closely allied agent, β, for instance, may need to balance this work with cooperative work with others over time. As α accumulates progress toward goals held in common with β (represented as an MQ), its preference may shift to the accumulation of other MQs. The use of utility for this application is flexible and very general and there are many different ways to relate organizational goal importance to the process of task valuation.

The model relates to other recent work in the multi-agent community, such as agents interacting via obligations [1], or notions of social commitment [4], but it differs in its quantification of different concerns and its dynamic, contextual, relative, evaluation of these. The model resembles MarCon [32] as the different degrees-of-satisfaction afforded by the MQ model is akin to MarCon's constraint optimization approach, and MarCon too deals with utilities/motivations that cannot always be commingled. MarCon, however, views constraints as agents, assigning particular roles to particular agents, and the issue of which tasks to perform do not enter into the problem space.

In the sections that follow we discuss scheduling MQ tasks and present examples of using the framework for agent control.

3 Scheduling and Analysis

If the agent's objective is to simply select which task to perform next, and tasks do not have associated deadlines or earliest start times, and the present and future value of MQs are equivalent, then it can reason using the maximum expected utility principle and select the task at each point that maximizes immediate utility. However, this simple choose-between-available-tasks model does not map well to situations in which tasks have individual earliest start times and/or deadlines. Note that in general, to coordinate the activities of multiple agents, temporal constraints such as these are needed to sequence activities over inter-agent interactions. In situations with such temporal interactions, it is difficult to produce optimal or even "good" results without considering sequences of activities and thus for most applications, scheduling of MQ tasks is required.

The implemented MQ task scheduler employs a generative state space search process where states record the agent's current MQ levels, utility functions, organizational roles and objectives, completed task set, candidate task set, and some estimation of the agent's future candidate task set. Transitions correspond to the performance of an MQ task. During the duration required for task performance (a transition), new tasks may arrive or the agent's organizational situation may change thus transitions can also be viewed as marking these changes as well. The MQ model was designed specifically to lend itself to this form of representation to ease reproducibility and to leverage the large body of research pertaining to state-based search. For example, the MQ model can be scheduled using standard real-time search technologies like anytime A^{s}tar [17], RTA* [25], and others [21,35], enabling the model to address the real-time requirement of online control for agents in open environments. This is particularly important because the search space is exponential in the number of actions being scheduled. Implementationally, the scheduler is unable to schedule problem instances of nine or more activities using exhaustive search[5] and other techniques are required. Depending on the characteristics of the problem instance, standard A* may produce results in a reasonable amount of time on a larger problem instance, though, approximate rather than admissible heuristics are sometimes required to avoid exhaustive generation by A*. Scheduling issues beyond the scope of this paper include approaches for constructing good search heuristics in the face of non-linear utility curves and considering the future value of MQs when estimating state potentials.

Opportunity cost also factors-into the scheduling process and into the estimation of the potential value of a given state. There are many different uses of opportunity cost in control and many different ways to compute or predict the future value of a unit of time. In the examples presented in this paper, the scheduler is configured so that opportunity cost is used to determine the value of a given activity relative to the time it requires to perform and it is computed using a running average of the amount of utility produced by the agent per time unit. Opportunity cost is tracked and expressed as a pair $< OCM, Window >$ where: 1) the OCM or *opportunity cost metric* expresses the current value of a unit of the agent's time, i.e., a utility-per-time-unit factor, and 2) the $Window$ denotes the time over which the OCM was produced. The $Window$ is necessary as time moves forward so the agent can reweight and adjust the OCM based on recent changes. As defined, opportunity cost is computed from the agent's initialization forward and will gradually be prone to little movement as weight ($Window$) is accumulated. There are obviously many different variations for the running average computation. Once the value of a unit of time is computed, the issue then becomes how to employ it when considering a given MQ task or estimating the potential of MQ tasks for use in search heuristics. In this paper the opportunity cost of an MQ task will have the same weight as the utility produced by the task. If t_i is a task being evaluated: $adjusted\_utility_{t_i} = utility_{t_i} - opportunity\_cost_{t_i}$. Note that initially the agent's OCM is 0 and its $Window$ is also 0, thus, in many cases the pair is "seeded" with initial estimations of appropriate values.

4 Demonstrating Control via MQs

In this section we demonstrate the use of the MQ model and the MQ task scheduler in an organizational control problem. The agent's objective in this example is to maximize its total utility over the set of candidate tasks. The agent in question is an *Information Gathering* agent for the Merrill Lynch corporation, IG_{ML}, and is situated in a network of financial information agents. The agent network is patterned after the the WARREN [6] style and the space is populated by three types of agents 1) *Database Manager* agents, 2) *Information Gathering* (IG) agents that are experts in particular domains and whose task is to plan, gather information, assimilate it, and produce a report, possibly accompanied by a recommendation to the client about a particular action to take based on

[5] On Pentium III class machine with 256 megabytes of RAM running Redhat Linux 6.0 and IBM's 1.1.6 jdk.

Organizational Membership IG_{ML} belongs to a single organization (Merrill Lynch).

Roles IG_{ML} has two different organizational roles and all tasks performed by IG_{ML} pertain to one role or the other:

1. IG_{ML} has a *service role* that requires servicing requests and queries from other agents seeking information. In this example, IG_{ML} will service requests from PA_{ML1} and PA_{ML2}.
2. IG_{ML} also has a *maintenance role* that entails keeping its data repository up to date so that it may effectively service requests. In this role, IG_{ML} performs update tasks and exploration tasks. Update tasks entail verifying the integrity of its database by confirming that it has valid and current information on the known database management agents. Exploration tasks entail seeking out new database management agents and building profiles of such sources for inclusion into its database.

Organizational Goals Produce profit by servicing requests. Maintain quality of existing repository to ensure long-term revenue stability. Grow repository to improve coverage and to keep pace with the growth of networked information sources. Repository growth is considered particularly valuable at this time.

Organizational Objectives The organizational goals translate into several organizational objectives for IG_{ML}.

1. During any period (we will explore one period), IG_{ML} is to give preference to maintenance tasks until it has performed a specified amount and then it is to balance request service with maintenance on the basis of returns.
2. Requests from PA_{ML1} are *preferred* to PA_{ML2} as they have a higher potential to produce more revenues for the company. IG_{ML} is thus advised to regard one unit of MQ from PA_{ML1} as having approximately twice the value of one unit of $MQ_{PA_{ML2}}$. (In this scenario there is no strict power relationship between the agents.)
3. Exploration tasks contribute more to the maintenance requirement than update tasks as it is perceived that growth is currently necessary. However, exploration tasks require more time to perform.

Fig. 1. IG_{ML}'s Organizational Context

the gathered information. 3) *Personal Agents* (PA) that interface directly with the human client, perhaps modeling the client's needs. These agents also decide with which information specialists to interact to solve a client's information need. In this paper, we focus on the interactions between IG_{ML} and personal agents for Merrill Lynch, PA_{ML1} and PA_{ML2}, that are associated with different Merrill Lynch employees. PA_{ML1} represents a mutual fund manager and PA_{ML2} represents an individual broker, thus, their requests must be evaluated differently by IG_{ML}.

In this scenario, IG_{ML} has the organizational roles and objectives shown in Figure 1. To track contributions toward each of its organizational roles, and thus to monitor the state of its requirement to perform a certain amount of maintenance prior to servicing data requests, IG_{ML} will use two different MQs: $MQ_{service}$ and $MQ_{maintenance}$. All tasks will produce some quantity of each of these MQs in addition to producing an MQ related to the task itself. IG_{ML} monitors and tracks the following MQs: MQ_{update}, $MQ_{explore}$, MQ_{PAML1}, MQ_{PAML2}, $MQ_{service}$, and $MQ_{maintenance}$. In accordance with the organizational objectives, the curve used for $MQ_{PA_{ML1}}$is $U = 2x + 2$ and for $MQ_{PA_{ML2}}$is $U = x + 1$. MQ_{update} and $MQ_{explore}$ are both assigned a curve of $U = 2x + 2$, though exploration tasks produce more units of $MQ_{maintenance}$ per the objectives. We model the maintenance versus service objective by using a curve with two segments for $MQ_{maintenance}$: $U = 2 * x \ when \ x <= 5$ and $U = x/2 \ when \ x > 5$ (the specified quantity of maintenance MQs to produce before servicing requests is 5 in this example). For $MQ_{service}$, we use a constant curve of $U = x/2$. In all cases, $\omega(U_i) = 1$. In this scenario, tasks do not produce unit quantities of MQs but instead produce different volumes of MQs.

```
Task: Update_1
    Alternative: Update_1.0
      Duration: 3.0
      MQPS u MQCS: {(mq_maintenance 2.5), (mq_update 1.5)}
      MQPS: {(mq_maintenance 2.5), (mq_update 1.5)}
      MQCS: {}

Task: Explore_1
    Alternative: Explore_1.0
      Duration: 10.0
      MQPS u MQCS: {(mq_maintenance 5.0), (mq_explore 2.0)}
      MQPS: {(mq_maintenance 5.0), (mq_explore 2.0)}
      MQCS: {}

Task: Update_2
    Alternative: Update_2.0
      Duration: 3.0
      MQPS u MQCS: {(mq_maintenance 2.5), (mq_update 1.5)}
      MQPS: {(mq_maintenance 2.5), (mq_update 1.5)}
      MQCS: {}

Task: Explore_2
    Alternative: Explore_2.0
      Duration: 10.0
      MQPS u MQCS: {(mq_maintenance 5.0), (mq_explore 2.0)}
      MQPS: {(mq_maintenance 5.0), (mq_explore 2.0)}
      MQCS: {}
```

```
Task: Request1_PAML1
    Alternative: Request1_PAML1.0
      Duration: 4.5
      MQPS u MQCS: {(mq_service 1.0), (mq_paml1 4.0)}
      MQPS: {(mq_service 1.0), (mq_paml1 4.0)}
      MQCS: {}

Task: Request2_PAML1
    Alternative: Request2_PAML1.0
      Duration: 4.5
      MQPS u MQCS: {(mq_service 1.0), (mq_paml1 4.0)}
      MQPS: {(mq_service 1.0), (mq_paml1 4.0)}
      MQCS: {}

Task: Request1_PAML2
    Alternative: Request1_PAML2.0
      Duration: 2.0
      MQPS u MQCS: {(mq_service 1.0), (mq_paml2 5.0)}
      MQPS: {(mq_service 1.0), (mq_paml2 5.0)}
      MQCS: {}

Task: Request2_PAML2
    Alternative: Request2_PAML2.0
      Duration: 2.0
      MQPS u MQCS: {(mq_service 1.0), (mq_paml2 5.0)}
      MQPS: {(mq_service 1.0), (mq_paml2 5.0)}
      MQCS: {}
```

Fig. 2. Service Requests and Maintenance Tasks

We will explore multiple variations using this model. First, we demonstrate appropriate agent control behavior given the organizational objectives. Consider the subset of IG_{ML}'s tasks and requests pictured in Figure 2. The tasks in the figure represent one half of the tasks assigned to the agent; there are actually two identical tasks of each task instance assigned to IG_{ML}, i.e., two requests from PA_{ML1}, two requests from PA_{ML2}, two exploration maintenance tasks and two updating maintenance tasks. Note that each of the tasks produce an MQ unique to the task type as well as an MQ relating to its broader organizational categorization (service or maintenance). The optimal schedule for IG_{ML} and the associated changes to the agent's state after each task is performed is shown in Figure 3 (the alternative identifier is omitted because each task has a single alternative).

In accordance with the specified objectives, the agent first elects to perform one of the exploration tasks. This produces the required five units of $MQ_{maintenance}$ as well as two units of $MQ_{explore}$. Because the utility curve for $MQ_{maintenance}$ changes after five units are produced, the agent then elects

	Task or Request Scheduled								
Attributes & State	Init	Explore2	Request2 PAML1	Request1 PAML1	Explore1	Request2 PAML2	Request1 PAML2	Update2	Update1
# in Sequence	n/a	1	2	3	4	5	6	7	8
Start Time	n/a	0	10	14.5	19	29	31	33	36
End Time	n/a	10	14.5	19	29	31	33	36	39
Utility After	7	21	29.5	38	44.5	50	55.5	59.75	64
Level MQ_{PAML1}	0	0	4	8	8	8	8	8	8
Level MQ_{PAML2}	0	0	0	0	0	5	10	10	10
Level MQ_{update}	0	0	0	0	0	0	0	1.5	3
Level $MQ_{explore}$	0	2	2	2	4	4	4	4	4
Level $MQ_{service}$	0	0	1	2	2	3	4	4	4
Level $MQ_{maintenance}$	0	5	5	5	10	10	10	12.5	15

Fig. 3. Optimal Schedule and Control State Changes for IG_{ML} That Balances Maintenance and Service

to service requests for PA_{ML1} rather than performing the remaining exploration task. Subsequent effort returns to the remaining exploration maintenance task, then the service of requests for PA_{ML2}, and finally the update tasks are performed. After the initial exploration task, and the change in $U_{maintenance}$, the choice between the exploration task, the two update tasks, and the requests for $PA_{ML1}$is determined by the quantity of MQs produced by each as $U_{maintenance} == U_{service}$ and $U_{explore} == U_{PAML1}$. The requests for PA_{ML2} are competitive with the update tasks for this same reason – though U_{PAML2} is dominated by U_{update} the requests for PA_{ML2} produce larger quantities of MQs than the update tasks.

Consider what happens if the organizational objective is changed and the maintenance objective is relaxed. In this case, the curve for the maintenance MQs is the same as that used for the service MQs, namely $U = x/2$ at all times. The optimal schedule and the state changes for the agent are shown in Figure 4. In this situation, the utility produced by requests for PA_{ML1} outweighs the benefits of the exploration tasks and they are performed first, rather than second. The exploration tasks are performed second, and the requests for PA_{ML2} follow, and finally the update tasks are performed. When the organizational objective is removed, the resulting task / utility curve set produces a non-interleaved ordering for the tasks (as each task of each type has the same MQ levels). This underscores the role of the two-segment utility curve modeling technique of the previous example in mapping the organizational objective into utility for the first 5 units of $MQ_{maintenance}$.

Consider a different scenario. Figure 5 shows the optimal schedule produced if we reinstate the organizational objective to produce 5 units of $MQ_{maintenance}$ before servicing requests, and, if we instruct the agent to factor-in the opportunity cost of the associated tasks. The agent is given an initial opportunity cost pair of $< OCM = 1.0, Window = 100.0 >$ In this case, the agent elects to satisfy the maintenance requirement by performing both of the update tasks, each of which produces 2.5 units of $MQ_{maintenance}$, rather than performing a single exploration task (5 units of $MQ_{maintenance}$ are produced by a single exploration task). This is because the exploration tasks require ten time units to execute and the update tasks require only three time units to execute. The utility state after each task without considering the opportunity cost of the task is given by

Attributes & State	Task or Request Scheduled								
	Init	Request2 PAML1	Request1 PAML1	Explore2	Explore1	Request2 PAML2	Request 1 PAML2	Update2	Update1
# in Sequence	n/a	1	2	3	4	5	6	7	8
Start Time	n/a	0	4.5	9	19	29	31	33	36
End Time	n/a	4.5	9	19	29	31	33	36	39
Utility After	7	15.5	24	30.5	37	42	48	52.25	56.5
Level MQ_{PAML1}	0	4	8	8	8	8	8	8	8
Level MQ_{PAML2}	0	0	0	0	0	5	10	10	10
Level MQ_{update}	0	0	0	0	0	0	0	1.5	3
Level $MQ_{explore}$	0	0	0	2	4	4	4	4	4
Level $MQ_{service}$	0	1	2	2	2	3	4	4	4
Level $MQ_{maintenance}$	0	0	0	5	10	10	10	12.5	15

Fig. 4. Optimal Schedule and Control State Changes for IG_{ML} When Maintenance Objective is Relaxed

Attributes & State	Task or Request Scheduled								
	Init	Update2	Update1	Request2 PAML1	Request1 PAML1	Request2 PAML2	Request1 PAML2	Explore2	Explore 1
# in Sequence	n/a	1	2	3	4	5	6	7	8
Start Time	n/a	0	3	6	10.5	15	17	19	29
End Time	n/a	3	6	10.5	15	17	19	29	39
Utility After	7	15	23	31.5	40	45.5	51	57.5	64
Utility$_{OC}$ After	7	12	˜ 16.8	˜ 20.4	˜ 23.8	˜ 27	˜ **31.2**	˜ **24.6**	˜ **19.4**
Level MQ_{PAML1}	0	0	0	4	8	8	8	8	8
Level MQ_{PAML2}	0	0	0	0	0	5	10	10	10
Level MQ_{update}	0	1.5	3	3	3	3	3	3	3
Level $MQ_{explore}$	0	0	0	0	0	0	0	2	4
Level $MQ_{service}$	0	0	0	1	2	3	4	4	4
Level $MQ_{maintenance}$	0	2.5	5	5	5	5	5	10	15

Fig. 5. Optimal Schedule and Control State Changes for IG_{ML} When Opportunity Cost is Considered

Utility After whereas the utility adjusted to reflect opportunity cost is indicated by *Utility$_{oc}$ After*. Note that the exploration tasks are the least appealing options to the agent and are scheduled last in this scenario. Note also that the agent's *Utility$_{oc}$* actually decreases when the exploration tasks are performed. This is because the tasks' opportunity costs are greater than the utility they produce. Depending on how the agent is configured, it may elect to wait-and-see while adjusting its opportunity cost OCM and $Window$ (as time moves forward) rather than performing the exploration tasks. In this case, the agent is configured to keep performing requests regardless. The first exploration task causes a net change in utility of -6.6 while the second exploration task incurs a lesser change of -5.2. This is because after performing the first exploration action, the agent's OCM is lowered by the negative utility produced by the task so that the opportunity cost of the second exploration task is less.

5 Conclusion, Limitations and Future Work

We have presented and extended the MQ model for local agent control of organized agents and shown its use in different applications. The strength of the model is its use of state to obtain appropriate local control – the model does not require common social-level control assumptions like shared or visible utility functions, which are useful in applications where shared and static knowledge are possible or as a theoretical foundation, e.g., [18].

Inherent in the model's state-based design are the assumptions that: 1) agents have imperfect knowledge of the problem solving taking place at other agents; 2) the utility function of a given agent cannot generally be shared and computed by other agents because it is dependent on the agent's problem solving state; 3) globally optimal behavior can be approximated through local reasoning. In this latter case, the precision of the approximation is dependent on the degree to which agents can *communicate* or *observe* progress toward organizational objectives. Consider IG_{ML}'s maintenance requirement from the previous section. If the maintenance requirement could be met by another IG agent of Merrill Lynch, e.g., IG_{ML_B}, and IG_{ML_B} decided to perform the required maintenance operations, the $MQ_{maintenance}$ requirement of both IG_{ML} and IG_{ML_B} would be met by IG_{ML_B}'s operation. However, IG_{ML}'s recognition of this depends on communication between the agents, observation, default reasoning, plan inference, or a similar mechanism. The communications required are beyond the scope of the MQ framework though the framework is designed explicitly to support such activities. Using the MQ model, notions of social utility are decomposed and distiled into the control regime of local agents – a feature we believe is important for application in large-scale MAS.

Intellectually, one of the contributions of the model is the attempt to address complexity in MAS – complexity that is even more important as we move to large-scale MAS and persistent agents. Agents in complex environments, having multiple organizational objectives and different relationships with other agents, require a certain level of complexity in their objective functions and in their action and situation models.

Another important characteristic of the framework is its support of local approximation of the global optimization problem of a large group of interacting agents. Aspects of this include the idea that different activities contribute to different aspects of the global objectives, the need to consider of history or state in decision making, and that in certain situations there are interactions between the local utility computations of different agents.

To summarize the model's positive attributes: 1) it enables organizationally appropriate behavior through local agent reasoning, 2) it is amenable to real-time control problem solving and this problem solving is fairly straightforward to reproduce as a state-based search, 3) the model is well suited to adjustable degrees of approximation – it can be optimal in a non-local sense when complete information is available and it can be very coarse when agent's have little information about the activities of other agents, 4) the model provides a unified evaluation framework for agent activities, 5) it represents aspects of the complexity inherent in large MAS.

In terms of problems and limitations, one attribute of the model that has not been explored fully is use of MQs as a medium of exchange. Initial explorations are in [51]. The most significant criticism of the model is that the translation of organizational structure into MQs, utility curves, initial MQ assignments, etc., as presented in this paper is ad-hoc. Though it required little knowledge engineering to produce the desired control behavior, experiments with a randomized assignment of MQ quantities to tasks illustrate a potential problem. Without design principles to guide the mapping, or appropriate corresponding verification techniques, it is difficult to be confident that a given mapping will result in the desired control behavior in the local agents. Ideally, the mapping of organizational objectives to organizational roles, the assignment of roles to agents, and the decomposition into MQs should be automated or performed by an organizational design component. Given the complexity of the problem, design principles and an approach for verification are the natural next step.

References

1. Mihai Barbuceanu. Agents that work in harmony by knowing and fulfiling their obligations. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 89–96, 1998.
2. Sviatoslav Brainov. The role and the impact of preferences on multiagent interaction. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI*, Lecture Notes in AI. Springer-Verlag, Berlin, 2000.
3. K.M. Carley and M.J. Prietula, editors. *Computational Organization Theory.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.
4. Cristiano Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS95)*, pages 41–48, 1995.
5. Phillip R. Cohen, Adam Cheyer, Michelle Wang, and Soon Cheol Baeg. An open agent architecture. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 197–204. Morgan Kaufmann, 1998.
6. K. Decker, A. Pannu, K. Sycara, and M. Williamson. Designing behaviors for information agents. In *Proceedings of the 1st Intl. Conf. on Autonomous Agents*, pages 404–413, Marina del Rey, February 1997.
7. Keith Decker and Jinjiang Li. Coordinated hospital patient scheduling. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*, pages 104–111, 1998.
8. Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms.* PhD thesis, University of Massachusetts, 1995.
9. Keith S. Decker, Edmund H. Durfee, and Victor R. Lesser. Evaluating research in cooperative distributed problem solving. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Vol. II*, pages 485–519. Pitman Publishing Ltd., 1989. Also COINS Technical Report 88-89, University of Massachusetts, 1988.
10. Keith S. Decker and Victor R. Lesser. Designing a family of coordination algorithms. In *Proceedings of the Thirteenth International Workshop on Distributed AI*, pages 65–84, Seattle, WA, July 1994. AAAI Press Technical Report WS-94-02. Also UMass CS-TR-94-14. To appear, Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, AAAI Press, 1995.
11. C. Dellarocas and M. Klein. An experimental evaluation of domain-independent fault handling services in open multi-agent systems. In *Proceedings of the Fifth International Conference on Multi-Agent Systems (ICMAS2000)*, 2000.
12. Robert Doorenbos, Oren Etzioni, and Daniel Weld. A scalable comparision-shopping agent for the world-wide-web. In *Proceedings of the First International Conference on Autonomous Agents*, pages 39–48, Marina del Rey, California, February 1997.

13. Edmund H. Durfee and Victor R. Lesser. Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, August 1987.
14. Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, 36(11):1275–1291, November 1987.
15. P. Faratin, C. Sierra, and N. Jennings. Negotiation Decision Functions for Autonomous Agents. *International Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1997.
16. Alan J. Garvey. *Design-to-Time Real-Time Scheduling*. PhD thesis, University of Massachusetts at Amherst, Amherst, Massachusetts, February 1996.
17. E.A. Hansen, S. Zilberstein, and V.A. Danilchenko. Anytime Heuristic Search: First Results. Department of Computer Science Technical Report TR-1997-50, University of Massachusetts, 1997.
18. Lisa Hogg and Nick Jennings. Variable sociability in agent-based decision making. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI*, Lecture Notes in AI. Springer-Verlag, Berlin, 2000.
19. Bryan Horling, Regis Vincent, Roger Mailler, Jiaying Shen, Raphen Becker, Kyle Rawlins, and Victor Lesser. Distributed sensor network for real-time tracking. In *Proceedings of Autonomous Agent 2001*, 2001.
20. Michael N. Huhns and Munindar P. Singh. Agents and multiagent systems: Themes, approaches, and challenges. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 1–23. Morgan Kaufmann, 1998.
21. Toru Ishida. Real-time search for autonomous agents and multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 1(2):139–167, October 1998.
22. Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):8–38, 1998.
23. N.R. Jennings, J.M.Corera, L. Laresgoiti, E.H. Mamdani, F. Perriollat, P. Skarek, and L.Z. Varga. Using ARCHON to develop real-world dai applications for electricity transportation management and particle accelerator control. *IEEE Expert*, 1995. Special issue on real world applications of DAI systems.
24. Henry Kautz, Bart Selman, Michael Coeh, Steven Ketchpel, and Chris Ramming. An experiment in the design of software agents. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 125–130. Morgan Kaufmann, 1998.
25. Richard E. Korf. Depth-limited search for real-time problem solving. *The Journal of Real-Time Systems*, 2(1/2):7–24, 1990.
26. Victor Lesser, Michael Atighetchi, Bryan Horling, Brett Benyo, Anita Raja, Regis Vincent, Thomas Wagner, Ping Xuan, and Shelley XQ. Zhang. A Multi-Agent System for Intelligent Environment Control. In *Proceedings of the Third International Conference on Autonomous Agents (Agents99)*, 1999.
27. Victor Lesser, Bryan Horling, and et al. The TÆMS whitepaper / evolving specification. http://mas.cs.umass.edu/research/taems/white.
28. Victor Lesser, Bryan Horling, Frank Klassner, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. BIG: An agent for resource-bounded information gathering and decision making. *Artificial Intelligence*, 118(1-2):197–244, May 2000. Elsevier Science Publishing.
29. Victor Lesser, Bryan Horling, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. Sophisticated Information Gathering in a Marketplace of Information Providers. *IEEE Internet Computing*, 4(2):49–58, Mar/Apr 2000.

30. Victor R. Lesser. Reflections on the nature of multi-agent coordination and its implications for an agent architecture. *Autonomous Agents and Multi-Agent Systems*, 1(1):89–111, 1998.
31. Victor R. Lesser and Daniel D. Corkill. Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):81–96, January 1981.
32. H. Van Dyke Parunak, Allen Ward, and John Sauter. A Systematic Market Approach to Distributed Constraint Problems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*, 1998.
33. Michael J. Prietula, Kathleen M. Carley, and Les Gasser. A Computational Approach to Oganizations and Organizing. In Michael J. Prietula, Kathleen M. Carley, and Les Gasser, editors, *Simulating Organizations: Computational Models of Institutions and Groups*, pages xiv–xix. AAAI Press / MIT Press, 1998.
34. Anita Raja, Victor Lesser, and Thomas Wagner. Toward Robust Agent Control in Open Environments. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents2000)*, 2000.
35. S.J. Russell. Efficient memory-bounded search methods. In *ECAI 92: 10th European Conference on Artifical Intelligence*, pages 1–5, 1992.
36. Paul A. Samuelson and William D. Nordhaus. *Economics*. McGraw-Hill Book Company, 1989. 13th Edition.
37. Sandip Sen and Anish Biswas. Effects of misconception on reciprocative agents. In *Proceedings of the Second International Conference on Autonomous Agents (Agents98)*, pages 430–435, 1998.
38. Milind Tambe. Agent Architectures for Flexible, Practical Teamwork. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 22–28, July 1997.
39. Thomas A. Wagner. *Toward Quantified Control for Organizationally Situated Agents*. PhD thesis, University of Massachusetts at Amherst, Amherst, Massachusetts, February 2000.
40. Manuela Veloso, Peter Stone, and Kwun Han. The CMUnited-97 robotic soccer team: Perception and multiagent control. In *Proceedings of the Second International Conference on Autonomous Agents (Agents98)*, pages 78–85, 1998.
41. R. Vincent, B. Horling, V. Lesser, and T. Wagner. Implementing Soft Real-Time Agent Control. In *Proceedings of Autonomous Agents (Agents-2001)*, 2001.
42. Thomas Wagner, Brett Benyo, Victor Lesser, and Ping Xuan. Investigating Interactions Between Agent Conversations and Agent Control Components. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, Lecture Notes in Artificial Intelligence, pages 314–331. Springer-Verlag, Berlin, 2000.
43. Thomas Wagner, Alan Garvey, and Victor Lesser. Criteria-Directed Heuristic Task Scheduling. *International Journal of Approximate Reasoning, Special Issue on Scheduling*, 19(1-2):91–118, 1998. A version also available as UMASS CS TR-97-59.
44. Thomas Wagner and Victor Lesser. Motivational Quantities: State-based Control for Organizationally Situated Agents. Computer Science Technical Report TR-99-68, University of Massachusetts at Amherst, November 1999. Research abstract appears in the proceedings of the International Conference on Multi-Agent Systems (ICMAS) 2000.
45. Thomas Wagner and Victor Lesser. Relating quantified motivations for organizationally situated agents. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI (Proceedings of ATAL-99)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2000.

46. Thomas Wagner and Victor Lesser. Design-to-Criteria Scheduling: Real-Time Agent Control. In Wagner/Rana, editor, *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, LNCS. Springer-Verlag, 2001. Also appears in the 2000 AAAI Spring Symposium on Real-Time Systems and a version is available as University of Massachusetts Computer Science Technical Report TR-99-58.
47. Thomas Wagner and Victor Lesser. Organizational level control for real-time agents. In *Proceedings of Autonomous Agents (Agents-2001)*, 2001.
48. Thomas Wagner, John Phelps, Yuhui Qian, Erik Albert, and Glen Beane. A modified architecture for constructing real-time information gathering agents. In *Proceedings of Agent Oriented Information Systems*, 2001.
49. M.P. Wellmen, E.H. Durfee, and W.P. Birmingham. The digital library as community of information agents. *IEEE Expert*, June 1996.
50. Mary Zey. *Rational Choice Theory and Organizational Theory: A Critique.* Sage Publications, Thousand Oaks, CA 91320, 1998.
51. Shelley XQ Zhang, Victor Lesser, and Thomas Wagner. A proposed approach to sophisticated negotiation. In *Proceedings of AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, 2001.

On the Epistemic Feasibility of Plans in Multiagent Systems Specifications

Yves Lespérance

Department of Computer Science, York University
Toronto, ON, Canada M3J 1P3
lesperan@cs.yorku.ca

Abstract. This paper addresses the problem of ensuring that agents' plans are epistemically feasible in multiagent systems specifications. We propose some solutions within the Cognitive Agents Specification Language (CASL). We define a subjective execution construct **Subj** that causes the plan to be executed in terms of the agent's knowledge state, rather than in terms of the world state. The definition assumes that the agent does not do planning or lookahead, and chooses arbitrarily among the actions allowed by the plan. We also define another deliberative execution operator **Delib** for smarter agents that do planning. We show how these notions can be used to express whether a process is epistemically feasible for its agent(s) in several types of situations. More generally, the paper shows how a formalization of epistemic feasibility can be integrated with a transition-system semantics for an agent programming/specification language.

1 Introduction

In the last few years, various frameworks have been proposed to support the formal specification and verification of multiagent systems (MAS) [1,7,8,29]. We have been involved in the development of such a framework, the Cognitive Agents Specification Language (CASL) [25]. CASL combines ideas from agent theory and formal methods, resulting in an expressive specification language that can be used to model and verify complex MAS.

One problem with CASL and some other MAS specification frameworks is that they do not provide good ways of ensuring that agents' plans are *epistemically feasible*, i.e., that the agents have enough knowledge to be able to execute their plans. In a real multiagent setting, each agent's behavior is determined by its own mental attitudes, i.e., its knowledge, goals, etc. At each point in time, agents must select what action to do next based on their plans and the knowledge that they have about the system's state. However in CASL (and some other frameworks), the system's behavior is simply specified as a set of concurrent processes. These processes may refer to agents' mental states — CASL includes operators that model agents' knowledge and goals — but they don't have to. There is no requirement that the modeler specify which agent is executing a given process and that he ensure that the agent has the knowledge required to execute the process.

Consider the following example adapted from Moore [18]. We have an agent, *Robbie*, that wants to open a safe, but does not know the safe's combination. There is also a second agent, $Smartie$, that knows the safe's combination. If we take the system's CASL specification to be just the primitive action:

$$dial(Robbie, combination(Safe1), Safe1),$$

that is, $Robbie$ dials the safe's combination and $Smartie$ does nothing, then we have a process that is physically executable and must terminate in a situation where the safe is open.[1] This holds provided that an appropriate specification of the effects and (physical) preconditions of the $dial$ action and of the initial situation has been given. However, this process is not epistemically feasible because the agent does not know the safe's combination.[2] Such a process specification may be adequate if all we want is to identify a set of runs of a system. But it does not capture the internal control of the agents, how their behavior is determined by their own mental state.

The fact that in CASL, system processes are specified from an outside observer's (third-person) point of view can have advantages. In many cases, the internal control programs of the agents are not known. Sometimes, the modeler may only want a very partial model of the system capturing some scenarios of interest. In the case of simple purely reactive agents, the modeler may not want to ascribe mental attitudes to the agents. Also, natural events and processes are best specified objectively. However, this loose coupling between specification and system means that it is easy to write specifications that could not be executed by agents. Often, one would want to ensure that the specifications are epistemically feasible for the agents.

For our example, if we want to ensure that the process is epistemically feasible for the agents, we should use a specification more like the following:

$$\begin{array}{l} (\,\mathbf{KRef}(Robbie, combination(Safe1))?;\\ \;dial(Robbie, combination(Safe1), Safe1)\,)\\ \|\\ informRef(Smartie, Robbie, combination(Safe1)). \end{array}$$

Here, in the first concurrent process, $Robbie$ waits until it knows what the safe's combination is and then dials it, and in the second process $Smartie$ tells $Robbie$ what the combination is — $\delta_1 \| \delta_2$ represents the concurrent execution of δ_1 and δ_2. We might also want to require that each agent know that the preconditions of its actions are satisfied before it does them, e.g., that $Robbie$ know that it is possible for him to dial a combination on the safe. This sort of requirements has been studied in agent theory under the labels "knowledge prerequisites of action", "knowing how to execute a program", "ability to achieve a goal", "epistemic feasibility", etc. [18,19,28,3,12,15,16]. The modeler could

[1] Formally:

$$\begin{array}{l} \exists s\, Do(dial(Robbie, combination(Safe1), Safe1), S_0, s) \wedge \\ \forall s(Do(dial(Robbie, combination(Safe1), Safe1), S_0, s) \supset Open(Safe1, s)). \end{array}$$

The notation is explained in Section 3.

[2] $combination(Safe1)$ is a fluent whose value varies according to the agent's epistemic alternatives; the situation argument can be made explicit by writing $combination(Safe1, now)$; see Section 3.

explicitly include all these knowledge prerequisites in the process specification. But it would be better if there was a way to simply say that the first process is going to be *subjectively executed* by *Robbie* and the second process by $Smartie$, and to have all the knowledge prerequisites fall out automatically; something like the following:

$$\begin{aligned}
&system1 \stackrel{\text{def}}{=}\\
&\mathbf{Subj}(\mathit{Robbie}, \mathbf{KRef}(\mathit{Robbie}, \mathit{combination}(Safe1))?;\\
&\qquad\qquad\qquad \mathit{dial}(\mathit{Robbie}, \mathit{combination}(Safe1), Safe1)) \parallel\\
&\mathbf{Subj}(Smartie, informRef(Smartie, \mathit{Robbie}, \mathit{combination}(Safe1))).
\end{aligned}$$

In this process specification which we call $system1$, we use a new construct **Subj**(agt, δ), which means that the process specification δ is subjectively executed by agent agt, that is, that δ is executed by agt in terms of his knowledge state. Note that we could have made the example more realistic by having *Robbie* request $Smartie$ to inform him of the combination and having $Smartie$ respond to such requests, as in the examples of [25]; but here, we prefer to keep the example simple. We return it in Section 3.

In this paper, we explore these issues, and propose an account of subjective plan execution in CASL that ensures that the plan can be executed by the agent based on its knowledge state. Our account of basic subjective execution (**Subj**) assumes that the agent does not do planning or lookahead as it executes its program. We also develop an account of deliberative plan execution (**Delib**) for smarter agents that do planning/lookahead. These notions are defined on top of CASL's transition-system semantics. In fact, one of the paper's contributions is showing how a formalization of epistemic feasibility can be adapted for use with an agent programming/specification language with a transition-system semantics. Note that the paper focuses on developing a reasonable model of agenthood for use in producing better specifications of MAS. This model should later prove useful for obtaining more accurate formal semantics for agent programming languages that interleave sensing and communication with planning and plan execution, e.g., IndiGolog [5].

2 Overview of CASL

The Cognitive Agents Specification Language (CASL) [25] is a formal specification language for multiagent systems. It combines a theory of action [22,23] and mental states [24] based on the situation calculus [17] with ConGolog [4], a concurrent, nondeterministic programming language that has a formal semantics. The result is a specification language that contains a rich set of operators to facilitate the specification of *complex* multiagent systems. A CASL specification of a system involves two components: a specification of the dynamics of the domain and a specification of the behavior of the agents in the system. Let us describe how these components are modeled.

2.1 Modeling Domain Dynamics

The *domain dynamics* component of a CASL specification states what properties and relations are used to model the state of the system, what actions may be performed by the agents, what their preconditions and effects are, and what is known about the initial state of the system (the specification may be incomplete). The model can include a specification of the agents' mental states, i.e., what knowledge and goals they have, as well as of the dynamics of these mental states, i.e., how knowledge and goals are affected by communication actions (e.g., inform, request, cancel-request, etc.) and perception actions. This component is specified in a purely declarative way in the situation calculus [17].

Very briefly, the situation calculus is a language of predicate logic for representing dynamically changing worlds. In this language, a possible world history (a sequence of actions) is represented by a first order term called a *situation*. The constant S_0 denotes the initial situation and the term $do(\alpha, s)$ denotes the situation resulting from action α being performed in situation s. Relations (functions) that vary from situation to situation, called *fluents*, are represented by predicate (function) symbols that take a situation term as last argument. The special predicate $Poss(\alpha, s)$ is used to represent the fact that primitive action α is physically possible in situation s.

We use Reiter's solution to the frame problem, where effects axioms are compiled into successor state axioms [22,23]. Thus, for our example, the domain dynamics specification includes the successor state axiom:

$$\begin{aligned}&Open(x, do(a, s)) \equiv \\ &\qquad \exists agt, c(a = dial(agt, c, x) \land c = combination(x, s)) \lor Open(x, s),\end{aligned}$$

i.e., the safe x is open in the situation that results from action a being performed in situation s if and only if a is the action of dialing x's correct combination on x or if x was already open in situation s. We also have a successor state axiom for the $combination$ fluent, whose value is not affected by any action:

$$combination(x, do(a, s)) = c \equiv combination(x, s) = c.$$

The specification also includes the action precondition axiom:

$$Poss(dial(agt, c, x), s) \equiv True,$$

i.e., that the $dial$ action is always physically possible. We also specify the agent of the action by:

$$agent(dial(agt, c, x)) = agt.$$

Knowledge is represented by adapting a possible world semantics to the situation calculus [18,24]. The accessibility relation $K(agt, s', s)$ represents the fact that in situation s, the agent agt thinks that the world could be in situation s'. An agent knows that ϕ if and only if ϕ is true in all his K-accessible situations:

$$\mathbf{Know}(agt, \phi, s) \overset{\text{def}}{=} \forall s'(K(agt, s', s) \supset \phi[s']).$$

Here, $\phi[s]$ represents the formula obtained by substituting s for all instances of the special constant now; thus for e.g., $\mathbf{Know}(agt, Open(Safe1, now), s)$ is an abbreviation for $\forall s'(K(agt, s', s) \supset Open(Safe1, s'))$. Often, when no confusion is possible, we suppress now altogether and simply write for e.g., $\mathbf{Know}(agt, Open(Safe1), s)$.

We assume that K is reflexive, transitive, and euclidean, which ensures that what is known is true, and that the agents always know whether they know something (positive and negative introspection). We also use the abbreviations $\mathbf{KWhether}(agt, \phi, s) \overset{\text{def}}{=} \mathbf{Know}(agt, \phi, s) \vee \mathbf{Know}(\neg agt, \phi, s)$, i.e., agt knows whether ϕ holds in s and $\mathbf{KRef}(agt, \theta, s) \overset{\text{def}}{=} \exists t\, \mathbf{Know}(agt, t = \theta, s)$, i.e., agt knows who/what θ is.

In this paper, we handle the following types knowledge-producing actions: binary sensing actions, e.g., $sense_{Open(Safe1)}(agt)$, where the agent senses the truth-value the associated proposition, non-binary sensing actions, e.g., $read_{combination(Safe1)}(agt)$, where the agent senses the value the associated term, and two generic communication actions, $informWhether(agt_1, agt_2, \phi)$ where agent agt_1 informs agent agt_2 of the truth-value of the proposition ϕ, and $informRef(agt_1, agt_2, \theta)$, where agent agt_1 informs agent agt_2 of the value of the term θ.[3] Following [15], the information provided by a binary sensing action is specified using the predicate $SF(a, s)$, which holds if action a returns the binary sensing result 1 in situation s. For example, we might have an axiom:

$$SF(sense_{Open(Safe1)}(agt), s) \equiv Open(Safe1, s),$$

i.e., the action $sense_{Open(Safe1)}(agt)$ will tell agt whether $Safe1$ is open in the situation where it is performed. Similarly for non-binary sensing actions, we use the term $sff(a, s)$ to denote the sensing value returned by the action; for example, we might have:

$$sff(read_{combination(Safe1)}(agt), s) = combination(Safe1, s),$$

i.e., $read_{combination(Safe1)}(agt)$ tells agt the value of $Safe1$'s combination.

We specify the dynamics of knowledge with the following successor state axiom:

$$\begin{aligned}
&K(agt, s^*, do(a, s)) \equiv \\
&\exists s'[K(agt, s', s) \wedge s^* = do(a, s') \wedge Poss(a, s') \wedge \\
&\quad (BinarySensingAction(a) \wedge agent(a) = agt \supset (SF(a, s') \equiv SF(a, s))) \wedge \\
&\quad (NonBinarySensingAction(a) \wedge agent(a) = agt \supset sff(a, s') = sff(a, s)) \wedge \\
&\quad \forall informer, \phi(a = informWhether(informer, agt, \phi) \supset \phi[s'] = \phi[s]) \wedge \\
&\quad \forall informer, \theta(a = informRef(informer, agt, \theta) \supset \theta[s'] = \theta[s])].
\end{aligned}$$

This says that that after an action happens, every agent learns that it has happened. Moreover, if the action is a sensing action, the agent performing it acquires knowledge of the associated proposition or term. Furthermore, if the action involves someone informing agt of whether ϕ holds, then agt knows this afterwards, and if the action involves someone informing agt of who/what θ is, then agt knows it afterwards ($\theta[s]$ stands for θ with s substituted for now, similarly to $\phi[s]$) . The preconditions of the communication actions are defined by the following axioms:

$$Poss(informWhether(informer, agt, \phi), s) \equiv \mathbf{KWhether}(informer, \phi, s),$$

i.e., $informWhether$ is possible in situation s if and only if $informer$ knows whether ϕ holds, and

$$Poss(informRef(informer, agt, \theta), s) \equiv \mathbf{KRef}(informer, \theta, s),$$

[3] Since the action $informWhether$ takes a formula as argument, we must encode formulas as terms; see [4] for how this is done. For notational simplicity, we suppress this encoding and use formulas as terms directly.

i.e., $informRef$ is possible in situation s if and only if $informer$ knows who/what θ is. There are also axioms stating that the informer is the agent of these actions. Goals and requests are modeled in an analogous way; see [25] for details.

Thus, the dynamics of a domain can be specified in CASL using an action theory that includes the following kinds of axioms:

- initial state axioms, which describe the initial state of the domain and the initial mental states of the agents;
- action precondition axioms, one for each action, which characterize $Poss$;
- successor state axioms, one for each fluent;
- axioms that specify which actions are sensing actions and what fluents they sense, characterizing SF and sff;
- axioms that specify the agent of every action;
- unique names axioms for the actions;
- some domain-independent foundational axioms [10,23].

2.2 Modeling Agent Behavior

The second component of a CASL model is a specification of the *behavior of the agents* in the domain. Because we are interested in modeling domains involving complex processes, this component is specified procedurally. For this, we use the ConGolog programming/process description language, which provides the following rich set of constructs:

α,	primitive action
$\phi?$,	wait for a condition
$\delta_1; \delta_2$,	sequence
$\delta_1 \mid \delta_2$,	nondeterministic branch
$\pi\, x\, \delta$,	nondeterministic choice of argument
δ^*,	nondeterministic iteration
if ϕ **then** δ_1 **else** δ_2 **endIf**,	conditional
while ϕ **do** δ **endWhile**,	while loop
$\delta_1 \parallel \delta_2$,	concurrency with equal priority
$\delta_1 \rangle\rangle\, \delta_2$,	concurrency with δ_1 at a higher priority
$\delta^{\parallel}$,	concurrent iteration
$\langle\, \boldsymbol{x} : \phi \to \delta\, \rangle$,	interrupt
$p(\boldsymbol{\theta})$,	procedure call.

The semantics of the ConGolog process description language [4] is defined in terms of *transitions*, in the style of structural operational semantics [21,9]. A transition is a single step of computation, either a primitive action or testing whether a condition holds in the current situation. Two special predicates are introduced, $Final$ and $Trans$, where $Final(\delta, s)$ means that program δ may legally terminate in situation s, and where $Trans(\delta, s, \delta', s')$ means that program δ in situation s may legally execute one step, ending in situation s' with program δ' remaining. $Trans$ and $Final$ are characterized by axioms such as:

$$\begin{aligned}
&Trans(\alpha, s, \delta, s') \equiv Poss(\alpha[s], s) \wedge \delta = nil \wedge s' = do(\alpha[s], s),\\
&Final(\alpha, s) \equiv False,\\
&Trans([\delta_1; \delta_2], s, \delta, s') \equiv\\
&\quad Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta, s')\\
&\quad \vee \exists \delta'(\delta = (\delta'; \delta_2) \wedge Trans(\delta_1, s, \delta', s')),\\
&Final([\delta_1; \delta_2], s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s).
\end{aligned}$$

The first axiom says that a program involving a primitive action α may perform a transition in situation s provided that $\alpha[s]$ is possible in s, with the resulting situation being $do(\alpha[s], s)$ and the remaining program being the empty program nil ($\alpha[s]$ stands for α with s substituted for now, similarly to $\phi[s]$). The second axiom says that a program with a primitive action remaining can never be considered to have terminated. The third axiom says that one can perform a transition for a sequence by performing a transition for the first part, or by performing a transition for the second part provided that the first part has already terminated. The last axiom says that a sequence has terminated when both parts have terminated.[4]

The axioms for the other ConGolog constructs that we will use are as follows:

$$\begin{aligned}
&Trans(\phi?, s, \delta, s') \equiv \phi[s] \wedge \delta = nil \wedge s' = s,\\
&Final(\phi?, s) \equiv False,\\
&Trans((\delta_1 \| \delta_2), s, \delta, s') \equiv\\
&\quad \exists \delta_1'(\delta = (\delta_1' \| \delta_2) \wedge Trans(\delta_1, s, \delta_1', s')\\
&\quad \vee \exists \delta_2'(\delta = (\delta_1 \| \delta_2') \wedge Trans(\delta_2, s, \delta_2', s')),\\
&Final((\delta_1 \| \delta_2), s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s),\\
&Trans((\delta_1 | \delta_2), s, \delta, s') \equiv Trans(\delta_1, s, \delta, s') \vee Trans(\delta_2, s, \delta, s'),\\
&Final((\delta_1 | \delta_2), s) \equiv Final(\delta_1, s) \vee Final(\delta_2, s),\\
&Trans(\pi\, v\, \delta, s, \delta', s') \equiv \exists x\, Trans(\delta^v_x, s, \delta', s'),\\
&Final(\pi\, v\, \delta, s) \equiv \exists x\, Final(\delta^v_x, s),\\
&Trans(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf}, s, \delta, s') \equiv\\
&\quad \phi[s] \wedge Trans(\delta_1, s, \delta, s') \vee \neg\phi[s] \wedge Trans(\delta_2, s, \delta, s'),\\
&Final(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf}, s) \equiv\\
&\quad \phi[s] \wedge Final(\delta_1, s) \vee \neg\phi[s] \wedge Final(\delta_2, s).
\end{aligned}$$

Note that to handle recursive procedures, a more complex formalization must be used; see [4] for the details.

[4] Note that we use axioms rather than rules to specify $Trans$ and $Final$ because we want to support reasoning about possible executions of a system given an incomplete specification of the initial situation. Since these predicates take programs (that include test of formulas) as arguments, this requires encoding formulas and programs as terms; see [4] for the details. For notational simplicity, we suppress this encoding and use programs as terms directly.

The overall semantics of a ConGolog program is specified by the Do relation:

$$
\begin{aligned}
&Do(\delta, s, s') \overset{\text{def}}{=} \exists \delta' (Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')),\\
&Trans^*(\delta, s, \delta', s') \overset{\text{def}}{=} \forall T[\\
&\qquad \forall \delta_1, s_1\, T(\delta, s, \delta, s) \wedge\\
&\qquad \forall \delta, s(Trans(\delta_1, s_1, \delta_2, s_2) \wedge T(\delta_2, s_2, \delta_3, s_3) \supset T(\delta_1, s_1, \delta_3, s_3))\\
&\quad \supset T(\delta, s, \delta', s')]
\end{aligned}
$$

$Do(\delta, s, s')$ holds if and only if s' is a legal terminating situation of process δ started in situation s, i.e., a situation that can be reached by performing a sequence of transitions starting with program δ in situation s and where the program may legally terminate. $Trans^*$ is the reflexive transitive closure of the transition relation $Trans$.[5]

CASL's approach aims for a middle ground between purely intentional (i.e., mental attitudes-based) specifications of agents, which typically allow only very weak predictions about the behavior of agents to be made, and the usual kind of concurrent process specifications, which are too low-level, and don't model mental states at all. Because of its logical foundations, CASL can accommodate incompletely specified models, both in the sense that the initial state of the system is not completely specified, and in the sense that the processes involved are nondeterministic and may evolve in any number of ways.

The latest version of the CASL framework, which supports communication with encrypted speech acts and provides a simplified account of goals is described in [25]. That paper also describes how CASL was used to model a complex multiagent system for feature interaction resolution in telecommunication applications, a system that involves negotiating, autonomous agents with explicit goals. Earlier versions of CASL are described in [26,13], where the use of the formalism is illustrated with a simple meeting scheduling multiagent system example. A discussion of how the process modeling features of the framework can be used for requirements engineering appears in [11]; this paper also discusses simulation and verification tools that are being developed.

3 Subjective Execution

We define the subjective execution construct **Subj**(agt, δ) introduced in Section 1 as follows:

$$
\begin{aligned}
&Trans(\textbf{Subj}(agt, \delta), s, \gamma, s') \equiv \exists \delta' (\gamma = \textbf{Subj}(agt, \delta') \wedge\\
&\quad [\textbf{Know}(agt, Trans(\delta, now, \delta', now), s) \wedge s' = s \vee\\
&\quad \exists a(\textbf{Know}(agt, Trans(\delta, now, \delta', do(a, now)) \wedge agent(a) = agt, s)\\
&\qquad \wedge s' = do(a, s))]),\\
&Final(\textbf{Subj}(agt, \delta), s) \equiv \textbf{Know}(agt, Final(\delta, now), s).
\end{aligned}
$$

This means that when a program is executed subjectively, the system can make a transition only if the agent knows that it can make this transition, and if the transition involves a

[5] To define the relation properly, we use second-order logic. For automated reasoning, one could use a first-order version to prove some, but not all, consequences of the theory.

primitive action, then this action must be performed by the agent himself. A subjective execution may legally terminate only if the agent knows that it may.

Let's go back to the examples of Section 1. Assume that in the initial situation S_0, *Robbie* does not know what the safe's combination is but *Smartie* does; formally:

$$\neg\mathbf{KRef}(Robbie, combination(Safe1), S_0) \wedge \\ \mathbf{KRef}(Smartie, combination(Safe1), S_0).$$

Then, we can easily show that *Robbie* cannot open the by himself, that the program where he just dials the safe's combination is not subjectively executable:

$$\neg\exists s Do(\mathbf{Subj}(Robbie, dial(Robbie, combination(Safe1), Safe1)), S_0, s).$$

To see this, observe that *Robbie*'s not knowing the combination initially amounts to $\neg\exists c\forall s(K(Robbie, s, S_0) \supset combination(Safe1, s) = c)$. Thus, there are K-accessible situations for *Robbie* in S_0, say s_1 and s_2, where $combination(Safe1, s_1) \neq combination(Safe1, s_2)$. It follows by the unique name axioms for actions that *Robbie*'s dialing the combination is a different action in these situations, i.e., $dial(Robbie, combination(Safe1, s_1), Safe1) \neq dial(Robbie, combination(Safe1, s_2), Safe1)$, and thus that *Robbie* does not know what that action is in S_0, i.e.,

$$\neg\exists a\,\mathbf{Know}(Robbie, dial(Robbie, combination(Safe1, now), Safe1), S_0).$$

Thus, *Robbie* does not know what transition to perform in S_0, i.e.,

$$\neg\exists a, \delta\,\mathbf{Know}(Robbie, Trans(DC, now, \delta, do(a, now), S_0),$$

where $DC \overset{\text{def}}{=} dial(Robbie, combination(Safe1, now), Safe1)$, and there is no transition where the program is subjectively executed, i.e., $\neg\exists\delta, s\, Trans(\mathbf{Subj}(Robbie, DC), S_0, \delta, s)$. Since the program cannot legally terminate (i.e., is not $Final$ in S_0), the program is not subjectively executable.

If on the other hand *Smartie* does inform *Robbie* of what the combination is, as in the $system1$ example, then one can easily show that the processes involved are subjectively executable, i.e., $\exists s Do(system1, S_0, s)$. Note that the test action $\mathbf{KRef}(Robbie, combination(Safe1))?$ in $system1$ is redundant, since as seen earlier, the $dial$ action cannot make a transition unless *Robbie* knows the combination.

The account also handles cases involving sensing actions. For example, if *Robbie* first reads the combination of the safe (assume he has it written on a piece of paper), and then dials it, then the resulting process is subjectively executable, i.e.:

$$\exists s Do(\mathbf{Subj}(Robbie, [read_{combination(Safe1)}(Robbie); \\ dial(Robbie, combination(Safe1), Safe1)]), S_0, s).$$

To get a better understanding of this notion, let's look at some of its properties. First, for a primitive action α, an agent can subjectively execute the action if it knows what the action is (including the value of fluent arguments such as $combination(Safe1)$) and knows that it is possible for him to execute it in the current situation:

Proposition 1.

$$\begin{aligned}&Trans(\textbf{Subj}(agt,\alpha),s,\delta,s') \equiv s' = do(\alpha[s],s) \wedge \delta = \textbf{Subj}(agt,nil) \wedge\\ &\quad \exists a \textbf{Know}(agt, \alpha = a \wedge Poss(a,now) \wedge agent(a) = agt, s).\end{aligned}$$

Secondly, for a test/wait action involving a condition ϕ, an agent can subjectively execute the action only if it knows that ϕ now holds:

Proposition 2.

$$Trans(\textbf{Subj}(agt,\phi?),s,\delta,s') \equiv s' = s \wedge \delta = \textbf{Subj}(agt,nil) \wedge \textbf{Know}(agt,\phi,s).$$

Thirdly, for an if-then-else program where the "then" and "else" branches involve different first transitions, an agent can subjectively execute it if it knows that the condition ϕ is now true and can subjectively execute the "then" branch, or knows that ϕ is now false and can subjectively execute the "else" branch:

Proposition 3.

$$\begin{aligned}&\neg\exists\delta,s'(Trans(\delta_1,s,\delta,s') \wedge Trans(\delta_2,s,\delta,s')) \supset\\ &\quad [Trans(\textbf{Subj}(agt, \textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf}, s,\delta,s') \equiv\\ &\qquad \textbf{Know}(agt,\phi,s) \wedge Trans(\textbf{Subj}(agt,\delta_1),s,\delta,s') \vee\\ &\qquad \textbf{Know}(agt,\neg\phi,s) \wedge Trans(\textbf{Subj}(agt,\delta_2),s,\delta,s')]\end{aligned}$$

So we see how with subjective execution, the tests and fluents that appear in the program as well as the preconditions of the actions, are all evaluated against the agent's knowledge state, rather than against the world state.

For deterministic single-agent programs, i.e., programs where $|$ (nondeterministic branch), π (nondeterministic choice of argument), $*$ (nondeterministic iteration), and $||$ (concurrency) do not occur, we can consider $\exists s' Do(\textbf{Subj}(agt,\delta),s,s')$ to be an adequate formalization of epistemic feasibility. In this case, it is sufficient that the agent know which transition to perform at each step and know when he can legally terminate. For nondeterministic programs on the other hand, we must consider how the agent chooses which transition to perform among the possibly many that are allowed. **Subj** can be viewed as capturing the behavior of an agent that executes its program in a bold or blind manner. When several transitions are allowed by the program, the agent chooses the next transition arbitrarily; it does not do any lookahead to make a good choice. So it can easily end up in a dead end. For example, consider the program $\textbf{Subj}(agt,(a;False?)|b)$, in a situation where the agent knows that both actions a and b are possible. Then, the agent might choose to do a transition by performing action a (rather than b), at which point the program $False?$ remains, for which there are no possible transitions and which cannot successfully terminate. If we want to ensure that an agent can subjectively execute a nondeterministic program successfully, we must ensure that every path through the program is subjectively executable and leads to successful termination. Note that this blind execution mode is the default one in the IndiGolog agent programming language [5].

So for nondeterministic programs, the existence of some path through the program (Do) that is subjectively executable is not sufficient to guarantee epistemic feasibility. We must ensure that all paths through the program are subjectively executable. We can capture this formally by defining a new predicate **AllDo**(δ, s) that holds if all executions of program δ starting in situation s eventually terminate successfully:

$$\begin{aligned}
&\textbf{AllDo}(\delta, s) \overset{\text{def}}{=} \forall R[\\
&\quad \forall \delta_1, s_1(Final(\delta_1, s_1) \supset R(\delta_1, s_1)) \wedge\\
&\quad \forall \delta_1, s_1(\exists \delta_2, s_2\, Trans(\delta_1, s_1, \delta_2, s_2) \wedge\\
&\qquad\qquad \forall \delta_2, s_2(Trans(\delta_1, s_1, \delta_2, s_2) \supset R(\delta_2, s_2))\\
&\qquad\qquad \supset R(\delta_1, s_1))\\
&\supset R(\delta, s)].
\end{aligned}$$

AllDo(δ, s) holds if and only if (δ, s) is in the least relation R such that (1) if (δ_1, s_1) can legally terminate, then it is in R, and (2) if some transition can be performed in (δ_1, s_1) and every such transition gets us to a configuration that is in R, then (δ_1, s_1) is also in R.[6] It is easy to see that if all executions of δ in s eventually terminate successfully then some execution eventually terminates successfully, i.e.:

$$\textbf{AllDo}(\delta, s) \supset \exists s' Do(\delta, s, s').$$

So, we formalize *epistemic feasibility* for single-agent programs where the agent executes the program blindly by the predicate **KnowHowSubj**(agt, δ, s), which is defined as follows:

$$\textbf{KnowHowSubj}(agt, \delta, s) \overset{\text{def}}{=} \textbf{AllDo}(\textbf{Subj}(agt, \delta), s)$$

i.e., every subjective execution of δ by agt starting in s eventually terminates successfully. For systems involving two agents agt_1 and agt_2 that blindly execute programs δ_1 and δ_2 concurrently, we can define epistemic feasibility as follows:

$$\textbf{KnowHowSubj}(agt_1, \delta_1, agt_2, \delta_2, , s) \overset{\text{def}}{=} \textbf{AllDo}(\textbf{Subj}(agt_1, \delta_1) \| \textbf{Subj}(agt_2, \delta_2), s).$$

Since the agents are not doing any lookahead, to guarantee that the process will be executed successfully, we must ensure that no matter how the agents' programs are interleaved and no matter which transitions they choose, the execution will terminate successfully. This can be generalized for processes that involve more than two agents and/or use composition methods other than concurrency. Again, if the agents are all executing their program blindly, then we must require that all executions terminate successfully. So for a multiagent process δ where agents are all blind executors — the agents' programs in δ must all be inside **Subj** operators — we define epistemic feasibility as follows:

$$\textbf{KnowHowSubj}(\delta, s) \overset{\text{def}}{=} \textbf{AllDo}(\delta, s).$$

Subj is very similar to the notion called "dumb knowing how" **DKH**(δ, s) formalized in [12] for δs that are Golog programs, i.e., ConGolog programs without concurrency or interrupts. For any deterministic single-agent Golog program δ, we believe that **Subj** and **DKH** are essentially equivalent, in the sense that:

$$\exists s' Do(\textbf{Subj}(agt, \delta), s, s') \equiv \textbf{DKH}(\delta, s).$$

6 **AllDo** is somewhat similar to the operator $\mathbf{AF}\phi$ in the branching time logic CTL^* [2]. Properties of processes like **AllDo** are often specified in the μ-calculus [20]; see [6] for a discussion in the context of the situation calculus.

We also believe that for nondeterministic single-agent Golog programs δ, we have that:

$$\textbf{AllDo}(\textbf{Subj}(agt, \delta), s) \equiv \textbf{DKH}(\delta, s).$$

(We hope to prove these conjectures in future work.) But note that **Subj** is considerably more general than **DKH**; **Subj** can be used to specify systems involving concurrent processes and multiple agents, as in the $system1$ example.

4 Deliberative Execution

In the previous section, we developed an account of subjective execution that took the agent to be executing the program blindly, without doing any lookahead or deliberation. In this section, we propose another account that captures when a smart agent that does deliberation knows how to execute a program. We use the notation $\textbf{Delib}(agt, \delta)$ for this notion of *deliberative execution*. It is formalized as follows:

$$\begin{array}{l}
Trans(\textbf{Delib}(agt,\delta), s, \gamma, s') \equiv \exists \delta'(\gamma = \textbf{Delib}(agt,\delta') \wedge \\
\quad [\textbf{Know}(agt, Trans(\delta, now, \delta', now) \wedge \textbf{KnowHowDelib}(agt, \delta', now), s) \wedge s' = s \vee \\
\quad \exists a(\textbf{Know}(agt, Trans(\delta, now, \delta', do(a, now)) \wedge agent(a) = agt \\
\qquad\qquad \wedge\, \textbf{KnowHowDelib}(agt, \delta', do(a, now)), s) \wedge s' = do(a, s))]),
\end{array}$$

$$\begin{array}{l}
\textbf{KnowHowDelib}(agt, \delta, s) \stackrel{\text{def}}{=} \forall R[\\
\quad \forall \delta_1, s_1(\textbf{Know}(agt, Final(\delta_1, now), s_1) \supset R(\delta_1, s_1)) \wedge \\
\quad \forall \delta_1, s_1(\exists \delta_2\, \textbf{Know}(agt, Trans(\delta_1, now, \delta_2, now) \wedge R(\delta_2, now), s) \\
\qquad\quad \supset R(\delta_1, s_1)) \wedge \\
\quad \forall \delta_1, s_1(\exists a, \delta_2\, \textbf{Know}(agt, Trans(\delta_1, now, \delta_2, do(a, now)) \wedge agent(a) = agt \\
\qquad\quad \wedge\, R(\delta_2, do(a, now)), s_1) \supset R(\delta_1, s_1)) \\
\supset R(\delta, s)],
\end{array}$$

$$Final(\textbf{Delib}(agt, \delta), s) \equiv \textbf{Know}(agt, Final(\delta, now), s).$$

This means that the system can make a transition only if the agent knows that it can make this transition and also knows how to deliberatively execute the rest of the program all the way to a final situation. The agent knows how to deliberatively execute δ in s, $\textbf{KnowHowDelib}(agt, \delta, s)$, if and only if (δ, s) is in the least relation R such that (1) if the agent knows that (δ_1, s_1) can legally terminate, then it is in R, and (2) if the agent knows that it can perform a transition in (δ_1, s_1) to get to a configuration that is in R, then (δ_1, s_1) is also in R. The system may legally terminate only if the agent knows that it may. We take $\textbf{KnowHowDelib}(agt, \delta, s)$ to be our formalization of epistemic feasibility for the case of single-agent (deterministic or nondeterministic) programs where the agent does deliberation/lookahead.

Let us look at an example. Consider the following program, which we call $system2$:

$$\begin{array}{l}
system2 \stackrel{\text{def}}{=} \\
\qquad \textbf{Subj}(Smartie, \pi\ c[dial(Smartie, c, Safe1); Open(Safe1)?]).
\end{array}$$

Here, $Smartie$ must nondeterministically choose a combination, dial it, and then verify that the safe is open. An agent that chooses transitions arbitrarily without doing lookahead is likely to choose and dial the wrong combination, since it does not consider the need to satisfy the test that the safe is open when it chooses its first transition. That is, we can show that:

$$\begin{aligned}&\exists \delta', c(c \neq \mathit{combination}(\mathit{Safe}1, S_0) \wedge \\ &\qquad \mathit{Trans}(\mathit{system}2, S_0, \delta', \mathit{do}(\mathit{dial}(\mathit{Smartie}, c, \mathit{Safe}1), S_0))).\end{aligned}$$

But a deliberative agent that does lookahead would be able to determine that it must dial the right combination. For $\mathit{system}2'$ which is just like $\mathit{system}2$, but with **Subj** replaced by **Delib**, the only possible transition is that where the agent dials the right combination. Thus, we can show that:

$$\begin{aligned}&\exists \delta', s'\, \mathit{Trans}(\mathit{system}2', S_0, \delta', s') \wedge \\ &\forall \delta', s'(\mathit{Trans}(\mathit{system}2', S_0, \delta', s') \supset \\ &\qquad s' = \mathit{do}(\mathit{dial}(\mathit{Smartie}, \mathit{combination}(\mathit{Safe}1), \mathit{Safe}1), S_0)).\end{aligned}$$

Let's look at some of the properties of **Delib** and compare it to **Subj**. First, we have that:

Proposition 4. *The properties of propositions 1, 2, and 3 also hold for* **Delib**.

To see where **Delib** and **Subj** differ, consider a program $\alpha; \delta$ involving a sequence that starts with a primitive action. Then, we have that:

Proposition 5.

$$\begin{aligned}&\mathit{Trans}(\mathbf{Subj}(\mathit{agt}, \alpha; \delta), s, \delta', s') \equiv s' = \mathit{do}(\alpha[s], s) \wedge \delta' = \mathbf{Subj}(\mathit{agt}, \mathit{nil}; \delta) \wedge \\ &\quad \exists a\, \mathbf{Know}(\mathit{agt}, \alpha = a \wedge \mathit{Poss}(a, \mathit{now}) \wedge \mathit{agent}(a) = \mathit{agt}, s) \quad \textit{and}\end{aligned}$$

$$\begin{aligned}&\mathit{Trans}(\mathbf{Delib}(\mathit{agt}, \alpha; \delta), s, \delta', s') \equiv s' = \mathit{do}(\alpha[s], s) \wedge \delta' = \mathbf{Delib}(\mathit{agt}, \mathit{nil}; \delta) \wedge \\ &\quad \exists a\, \mathbf{Know}(\mathit{agt}, \alpha = a \wedge \mathit{Poss}(a, \mathit{now}) \wedge \mathit{agent}(a) = \mathit{agt} \\ &\qquad\qquad \wedge \mathbf{KnowHowDelib}(\mathit{agt}, [\mathit{nil}; \delta], \mathit{do}(a, \mathit{now})), s).\end{aligned}$$

That is, with **Delib**, the agent must not only know what transition/action to perform next, but also know that he will know how to complete the execution of what remains of the program after that transition. Clearly, **Delib** puts much stronger requirements on the agent than **Subj**. The fact that we can count on an agent that deliberates to choose his transitions wisely means that for a nondeterministic program to be epistemically feasible, we no longer need to require that all executions lead to successful termination; it is sufficient that the agent know that no matter how his sensing actions turn out, he will be able to get to some final configuration of the program.

The deliberative execution mode modeled by **Delib** is similar to that used by the IndiGolog agent programming language [5] for programs that are enclosed in a "search block". It is also similar to a notion of ability called **Can**$_\perp$ that is formalized in [12]. Essentially, the agent must be able to construct a strategy (a sort of conditional plan) such that if the program is executed according to the strategy, the agent will know what action to perform at every step and be able to complete the program's execution in some bounded number of actions. In [12], we show that **Can**$_\perp$ is not as general as one might wish and that there are programs that one would intuitively think a deliberative agent can execute for which **Can**$_\perp$ does not hold. These are cases that involve indefinite iteration and where the agent knows that he will eventually complete the program's execution, but cannot bound the number of actions that have to be performed. [12] proposes an account of ability that also handles these cases. However, the type of deliberation required for this is hard to implement; the **Delib** account corresponds more closely to that implemented in IndiGolog [5].

For the multiagent case, we don't yet have a satisfactory general formalization of epistemic feasibility for agents that deliberate. The problem is that if the processes involve true interactions and the agents' choice of actions must depend on that of other agents, then each agent has to model the other agents' deliberation. Moreover, one must take into account whether the agents are cooperative, competitive, or indifferent. A simple example of this kind of system is one where $Robbie$ wants to open the safe and since it does not the combination is when it starts to deliberate, plans to ask $Smartie$ for it, expecting to get an answer. A mechanism to support a simple version of this kind of deliberation in IndiGolog has been proposed in [14] (where the deliberating agent models other agents as deterministic processes). We would like to extend our formalization to handle this and other cases.

5 Conclusion

In this paper, we have dealt with the problem of ensuring that agents' plans are epistemically feasible in formal specifications of multiagent systems. The problem is tied with that of capturing an adequate notion of agenthood where an agent's choice of actions is determined by its local state. The paper deals with this in the context of the CASL specification language, but the problem arises in other frameworks based on specifying multiagent systems as a set of concurrent processes. We have proposed an account of *subjective plan execution* (**Subj**) that ensures that the plan is executed in terms of what the agent's knows rather than in terms of is true in the world. This account assumes that the agent does not deliberate or do lookahead as it executes its plan. We have also proposed an account of *deliberative plan execution* (**Delib**) for smarter agents that do planning/lookahead. Finally, in terms of these, we have developed two formalizations of *epistemic feasiblilty:* one that captures whether any set of multiagent processes is epistemically feasible when the agents do not deliberate/do lookahead (**KnowHowSubj**), and another that captures whether a process involving a single agent is epistemically feasible where the agent does deliberate/do lookahead (**KnowHowDelib**). The case of multiagent processes where the agents deliberate remains open. Our formalization shows how an account of epistemic feasibility can be integrated with a transition-system semantics for an agent programming/specification language.

Let's examine where other multiagent systems specifications frameworks stand with respect to this problem of ensuring that plans are epistemically feasible. In van Eijk *et al.*'s MAS specification framework [29], which is inspired from standard concurrent systems programming formalisms and adds an account of agents' belief states, programs are executed in a subjective manner. Tests are evaluated in terms of the executing agent's beliefs and primitive actions are treated as belief update operations. However, there is no representation of the agents' external environment and it is not possible to model agents' interactions with their external environment (e.g., to talk about the reliability of their sensors or effectors). One can of course represent the environment as an agent, but this is not completely satisfactory; for instance, the world is always complete and never wrong. As well, there is no account of deliberative execution; agents are assumed not to do any lookahead in executing their plans.

The MAS specification framework of Engelfriet *et al.* [7] is a modal logic with temporal and epistemic modalities. It is harder to specify systems with complex behaviors in this type of formalism than in procedural languages like CASL or [29]. If agents are specified according to the proposed methodology, their choice of actions will depend only on their local state. But as in [29], agents don't do any lookahead and there is no account of deliberative execution. Engelfriet *et al.* [7] describe a compositional verification methodology based on their framework.

None of these papers really discuss the conditions under which plans are epistemically feasible. In the case of deterministic single-agent processes, epistemic feasibility reduces to the existence of a successful subjective execution, so both frameworks can be viewed as handling the problem. But there is no treatment for other cases. Neither framework supports the objective execution of processes (the default in CASL).

The work described in this paper is ongoing. In particular, we still need to make a more systematic comparison with earlier work on epistemic feasibility and to develop adequate definitions of epistemic feasibility for the cases of multiagent systems where agents deliberate. We also intend to use our account to produce a more accurate formal semantics for the IndiGolog agent implementation language [5,14], accounting for its on-line execution mechanism, where sensing, planning/deliberation, and plan execution are interleaved. We would also like examine how our account of subjective/deliberative execution can be adapted to the more general treatment of epistemic attitudes of [27], which allows false beliefs and supports their revision. This would allow us to model cases where the agent executes a process in a way that it thinks is correct, but that may in fact be incorrect. Cases of uninformed or insincere communication could also be handled.

Acknowledgments

I have greatly benefited from discussions on this topic with Hector Levesque, Giuseppe De Giacomo, Steven Shapiro, and David Tremaine. Comments from one of the referees were also very helpful. This research was funded by Communications and Information Technology Ontario and the Natural Science and Engineering Research Council of Canada.

References

1. F. Brazier, B. Dunin-Keplicz, N.R. Jennings, and Jan Treur. Formal specifications of multi-agents systems: A real-world case study. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 25–32, San Francisco, CA, June 1995. Springer-Verlag.
2. E. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Trans. Programming, Languages, and Systems*, 8(2):244–263, 1986.

3. Ernest Davis. Knowledge preconditions for plans. *Journal of Logic and Computation*, 4(5):721–766, 1994.
4. Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.
5. Giuseppe De Giacomo and Hector J. Levesque. An incremental interpreter for high-level programs with sensing. In Hector J. Levesque and Fiora Pirri, editors, *Logical Foundations for Cognitive Agents*, pages 86–102. Springer-Verlag, Berlin, Germany, 1999.
6. Giuseppe De Giacomo, Eugenia Ternovskaia, and Ray Reiter. Non-terminating processes in the situation calculus. In *Working Notes of the AAAI'97 Workshop on Robots, Softbots, Immobots: Theories of Action, Planning and Control*, 1997.
7. Joeri Engelfriet, Catholijn M. Jonker, and Jan Treur. Compositional verification of multi-agent systems in temporal multi-epistemic logic. In J.P. Mueller, M.P. Singh, and A.S. Rao, editors, *Intelligent Agents V: Proceedings of the Fifth International Workshop on Agent Theories, Architectures and languages (ATAL'98)*, volume 1555 of *LNAI*, pages 177–194. Springer-Verlag, 1999.
8. M. Fisher and M. Wooldridge. On the formal specification and verification of multi-agent systems. *International Journal of Cooperative Information Systems*, 6(1):37–65, 1997.
9. M. Hennessy. *The Semantics of Programming Languages*. John Wiley & Sons, 1990.
10. Gerhard Lakemeyer and Hector J. Levesque. AOL: A logic of acting, sensing, knowing, and only-knowing. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR-98)*, pages 316–327, 1998.
11. Yves Lespérance, Todd G. Kelley, John Mylopoulos, and Eric S.K. Yu. Modeling dynamic domains with ConGolog. In *Advanced Information Systems Engineering, 11th International Conference, CAiSE-99, Proceedings*, pages 365–380, Heidelberg, Germany, June 1999. LNCS 1626, Springer-Verlag.
12. Yves Lespérance, Hector J. Levesque, Fangzhen Lin, and Richard B. Scherl. Ability and knowing how in the situation calculus. *Studia Logica*, 66(1):165–186, October 2000.
13. Yves Lespérance, Hector J. Levesque, and Raymond Reiter. A situation calculus approach to modeling and programming agents. In A. Rao and M. Wooldridge, editors, *Foundations of Rational Agency*, pages 275–299. Kluwer, 1999.
14. Yves Lespérance and Ho-Kong Ng. Integrating planning into reactive high-level robot programs. In *Proceedings of the Second International Cognitive Robotics Workshop*, pages 49–54, Berlin, Germany, August 2000.
15. Hector J. Levesque. What is planning in the presence of sensing? In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1139–1146, Portland, OR, August 1996.
16. Fangzhen Lin and Hector J. Levesque. What robots can do: Robot programs and effective achievability. *Artificial Intelligence*, 101(1–2):201–226, 1998.
17. John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, UK, 1979.
18. Robert C. Moore. A formal theory of knowledge and action. In J. R. Hobbs and Robert C. Moore, editors, *Formal Theories of the Common Sense World*, pages 319–358. Ablex Publishing, Norwood, NJ, 1985.
19. Leora Morgenstern. Knowledge preconditions for actions and plans. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 867–874, Milan, Italy, August 1987. Morgan Kaufmann Publishing.
20. D. Park. Fixpoint induction and proofs of program properties. In *Machine Intelligence*, volume 5, pages 59–78. Edinburgh University Press, 1970.

21. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Computer Science Dept., Aarhus University, Denmark, 1981.
22. Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.
23. Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
24. Richard B. Scherl and Hector J. Levesque. The frame problem and knowledge-producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 689–695, Washington, DC, July 1993. AAAI Press/The MIT Press.
25. Steven Shapiro and Yves Lespérance. Modeling multiagent systems with CASL — a feature interaction resolution application. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII. Agent Theories, Architectures, and Languages — 7th. International Workshop, ATAL-2000, Boston, MA, USA, July 7–9, 2000, Proceedings*, volume 1986 of *LNAI*, pages 244–259. Springer-Verlag, Berlin, 2001.
26. Steven Shapiro, Yves Lespérance, and Hector J. Levesque. Specifying communicative multi-agent systems with ConGolog. In *Working Notes of the AAAI Fall 1997 Symposium on Communicative Action in Humans and Machines*, pages 75–82, Cambridge, MA, November 1997.
27. Steven Shapiro, Maurice Pagnucco, Yves Lespérance, and Hector J. Levesque. Iterated belief change in the situation calculus. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR-2000)*, pages 527–538. Morgan Kaufmann, 2000.
28. W. van der Hoek, B. van Linder, and J.-J. Ch. Meyer. A logic of capabilities. In A. Nerode and Yu. V. Matiyasevich, editors, *Proceedings of the Third International Symposium on the Logical Foundations of Computer Science (LFCS'94)*. LNCS Vol. 813, Springer-Verlag, 1994.
29. Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Open multi-agent systems: Agent communication and integration. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Lecture Notes in Artificial Intelligence, pages 218–232. Springer-Verlag, Berlin, 2000.

On Multi-agent Systems Specification via Deontic Logic

Alessio Lomuscio and Marek Sergot

Department of Computing
Imperial College of Science, Technology and Medicine
London SW7 2BZ, United Kingdom
{A.Lomuscio,M.Sergot}@doc.ic.ac.uk

Abstract. We investigate an extension of the formalism of interpreted systems by Halpern and colleagues to model correct behaviour of agents. The semantical model allows for the representation and reasoning about states of correct and incorrect functioning behaviour of the agents, and of the system as a whole. We axiomatise this semantic class by mapping it into a suitable class of Kripke models. The resulting logic, $\mathrm{KD45}_n^{i\text{-}j}$, is a stronger version of KD, the system often referred to as Standard Deontic Logic. We discuss these issues and present some simple examples.

1 Introduction

Formal methods and logic in particular have a long tradition in artificial intelligence and distributed computing. Their role, it is argued, is to provide a precise and unambiguous formal tool to *specify* and *reason about* complex systems. However, they have often been attacked by software engineers because of the allegedly somewhat unclear contribution they make towards the engineering of complex computing systems. One of the criticisms most often aired is that logic specifications do not provide *constructive methodologies* for building distributed systems, and so they can be of only limited significance in practice. These different views have led the fields of theoretical and practical distributed computing to diverge.

Somewhat unsurprisingly, the advent of the intelligent agents paradigm in distributed computing, together with the tremendous growth of the Internet medium, has widened this gap even further. On the one hand, the intrinsic complexity of the capabilities required by agents has encouraged theoreticians to consider logics for intensional concepts developed by philosophers; on the other hand, the commercialisation of the Internet together with the emphasis on timed product delivery from software houses has progressively led practitioners to ignore the results being produced by theorists of multi-agent systems (MAS).

Still, if MAS (see [18] for a recent introduction) are to fulfil the promises of providing the architectural foundations of increasingly critical and complex Internet applications, such as automatic negotiation in e-commerce, remote control of safety critical systems, etc, then there is clearly a need for bridging the gap between theory and practice. In our view, this is particularly relevant with respect to the issues relating to specification and verification of MAS.

One of the suggestions that have been put forward [17] (and that we fully endorse) to make MAS theories more relevant to practitioners is the shift to a semantics which is *computationally grounded*. This remark applies to distributed artificial intelligence in general but it is particularly relevant for the case of MAS theories, where semantics are usually given by borrowing ideas developed originally in philosophy. Indeed, most of the more highly respected theories for modelling knowledge, beliefs, intentions, obligations, communications, etc, in MAS are based upon works done in the second half of the last century in philosophical logic. While cross-fertilisation of fruitful ideas can only be regarded positively, one should note that the semantics developed in philosophical logic, even if appropriate for the original task, may not be the best option for distributed computing applications.

As is widely known, the semantics commonly used for MAS theories is based on Kripke models [8]. A Kripke model $M = (W, R_1, \dots, R_n, \pi)$ is a tuple composed of a set W, n relations $R_i \subseteq W \times W$, together with an interpretation π for the atoms of the language. The points represent possible alternatives of the world, and depending on the application under consideration, stand for temporal snapshots of the evolution of the world (temporal logic), epistemic alternatives (epistemic logic), and so on. Various modal operators can be interpreted by using this semantics, and a heritage of techniques has been developed to prove meta-logical properties about the logics. Notwithstanding this, no clear correspondence can be drawn between a Kripke model and a distributed computing system; in particular, it is a matter of debate whether or not a correspondence can be drawn between the notion of world in a Kripke model and that of state of a computational system.

It has been argued that this lack of correspondence is a serious drawback for attempts to close the gap between theory and practice. Indeed, without a relevant semantics well studied meta-logical properties such as completeness do not seem to be of relevance, and the only possible point of contact between the theorist and the practitioner seems to be logical formulas representing specifications that the theorist would propose to the practitioner. Appropriate grounded semantics aim at bridging this gap by providing practitioners and theoreticians with a convenient and intuitive semantical tool. A grounded semantics should aim at ensuring that a clear correspondence can, at least in principle, be found between states in the computing system and configurations in the semantical description.

The idea of moving away from Kripke models, while still benefiting from most of its technical apparatus, is not new. Indeed, part of the knowledge representation literature uses modal languages defined on semantic structures called *Interpreted systems* (see Chapter 4 of [2] for details). The idea is to describe a distributed computing system by specifying the states in which every agent and the environment can find itself. In this setting, the level of abstraction at which one chooses to operate is left quite open; one has the possibility of adopting a fine grain of detail by describing precisely the protocol that the MAS operates, or one can limit oneself to describing macroscopic properties of the MAS, such as epistemic and temporal properties. If needed, logics similar or equivalent to the ones used in philosophical logics can be defined on these semantics. One obvious advantage is the possibility of studying the logic characterisation of systems defined semantically, as opposed to isolating complete semantical structures for a specification.

In this paper we run the following exercise. We consider the basic notion of interpreted system as defined by Halpern et al. in [2] and show how it can be trivially adapted to provide a basic grounded formalism for some deontic issues. In particular we aim at representing local and global states of violation and compliance (with respect to some functioning protocol). By using these concepts we would like to give a grounded semantics to the deontic notions of *ideal functioning behaviour* of an agent, or of a system of agents. Once this task is accomplished, we axiomatise this resulting semantical class and draw correspondences between the obtained results and some well-known in the literature of deontic logic.

The rest of the paper is organised as follows. In the next section we fix the notation and point to some basic modal logic facts that will become useful later on. In Section 2, we define deontic interpreted systems, and define satisfaction, and validity, of a modal language on them. In Section 3 we study their axiomatisation. In Section 4 we comment on the results of the paper and draw attention to issues for further work. We conclude in Section 5.

2 Deontic Interpreted Systems

2.1 Syntax

We assume a set P of propositional atoms, and a set $A = 1, \ldots, n$ of agents.

Definition 1. *The language $\mathcal{L}$ is defined as follows.*

$$\varphi ::= \textbf{false} \mid \textit{any element of } P \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathcal{O}_i\, \varphi \qquad (i \in A).$$

We use the indexed modal operator $\mathcal{O}_i$ to represent the *correctly functioning circumstances of agent* i: the formula $\mathcal{O}_i\, \varphi$ stands for "in all the possible correctly functioning alternatives of agent i, φ is the case", or "whenever agent i is functioning correctly (with respect to some protocol or specification) φ is the case". The formula φ can either refer to local or global properties or to both at the same time. We write $\mathcal{P}_i$ for the dual of $\mathcal{O}_i$: $\mathcal{P}_i\, \varphi =_{def} \neg\, \mathcal{O}_i\, \neg\varphi$. Intuitively, $\mathcal{P}_i\, \varphi$ stands for "in some of the states in which agent i operates correctly φ holds", or "φ holds in some of the correct functioning alternatives of agent i". We have chosen the symbol $\mathcal{O}_i$ because its semantics will be similar to that of the obligation operator of standard deontic logic. However, it would not be appropriate to read $\mathcal{O}_i\, \varphi$ as "it is obligatory for agent i that φ".

Note. In line with much of the literature, we denote a normal modal logic by listing the axioms that define it, under the assumption that uniform substitution, necessitation, and modus ponens hold. For example by $\mathrm{KT45}_n$ we denote the normal logic obtained by considering axioms K, T, 4, and 5 for n agents (see [2] for more details). In the following we assume familiarity with basic modal logic techniques and results. We refer the reader to [7,13] for more details.

2.2 Deontic Interpreted Systems

Interpreted systems were originally defined by Halpern and Moses [5], and their potentiality later presented in greater detail in [2]. They provide a general framework for reasoning about properties of distributed systems, such as synchrony, a-synchrony, communication, failure properties of communication channels, etc. One of the reasons for the success of interpreted systems is the ease with which states of knowledge can be ascribed to the agents in the system.

The fundamental notion on which interpreted systems are defined is the one of 'local state'. Intuitively, the local state of an agent represents the entire information about the system that the agent has at its disposal. This may be as varied as to include program counters, variables, facts of a knowledge base, or indeed a history of these. The (instantaneous) state of the system is defined by taking the local states of each agent in the system, together with the local state for the environment. The latter is used to represent information which cannot be coded in the agents' local states such as messages in transit, etc.

More formally, consider n non-empty sets $L_1, \ldots, L_n$ of local states, one for every agent of the system, and a set of states for the environment L_e. Elements of L_i will be denoted by $l_1, l'_1, l_2, l'_2, \ldots$. Elements of L_e will be denoted by $l_e, l'_e, \ldots$.

Definition 2 (System of global states). *A* system of global states for n agents S *is a non-empty subset*[1] *of a Cartesian product* $L_e \times L_1 \times \cdots \times L_n$.

An interpreted *system of global states is a pair* (S, π) *where* S *is a system of global states and* $\pi : S \to 2^P$ *is an interpretation function for the atoms.*

The framework presented in [2] represents the temporal evolution of a system by means of *runs*; these are functions from the natural numbers to the set of global states. An *interpreted system*, in their terminology, is a set of runs over global states together with a valuation for the atoms of the language on points of these runs. In this paper we do not deal with time, and so we will simplify this notion by not considering runs.

We now define *deontic systems of global states* by assuming that for every agent, its set of local states can be divided into allowed and disallowed states. We indicate these as *green states*, and *red states* respectively.

Definition 3 (Deontic system of global states). *Given* n *agents and* $n+1$ *mutually disjoint and non-empty sets* $G_e, G_1, \ldots, G_n$, *a* deontic system of global states *is any system of global states defined on* $L_e \supseteq G_e, \ldots, L_n \supseteq G_n$. G_e *is called the* set of green states for the environment, *and for any agent* i, G_i *is called* the set of green states for agent i. *The complement of* G_e *with respect to* L_e *(respectively* G_i *with respect to* L_i*) is called the* set of red states for the environment (respectively for agent i*)*.

Given an agent, red and green local states respectively represent 'disallowed' and 'allowed' states of computation. An agent is in a disallowed state if this is in contravention of its specification, as is the case, for example, in a local system crash, or a memory violation. The notion is quite general however: classifying a state as 'disallowed' (red) could simply signify that it fails to satisfy some desirable property.

Note that any collection of red and green states as above identifies a *class* of global states. The class of deontic systems of global states is denoted by $\mathcal{DS}$.

[1] The case of the full Cartesian product was analysed in [9].

Definition 4 (Interpreted deontic system of global states). *An* interpreted deontic system of global states *IDS for n agents is a pair $IDS = (DS, \pi)$, where DS is a deontic system of global states, and π is an interpretation for the atoms.*

In the knowledge representation literature interpreted systems are used to ascribe knowledge to agents, by considering two global states to be indistinguishable for an agent if its local states are the same in the two global states. Effectively, this corresponds to generating a Kripke frame from a system of global states (some formal aspects of this mapping have been explored in [10]). In this case, the relations on the generated Kripke frame are equivalence relations; hence (see [15,2]) the logic resulting by defining a family of modal operators representing a 'bird's eye view' of the knowledge of the agents is $S5_n$.

In this paper we set out to do a similar exercise. We investigate how to axiomatize deontic systems of global states using the languages defined in Definition 1, and study the properties of the resulting formalisation. In the spirit of the interpreted systems literature we interpret modal formulas on the Kripke models that are built from deontic systems of global states. In order to do this, we first define the frame generated by a deontic system.

Definition 5 (Frame generated by a system). *Given a deontic system of global states DS, the* generated frame *$F(DS) = (W, R_1, \ldots, R_n)$ is defined as follows.*

- *$W = DS$.*
- *For any $i = 1, \ldots, n$, $\langle l_e, l_1, \ldots, l_n \rangle \, R_i \, \langle l'_e, l'_1, \ldots, l'_n \rangle$ if $l'_i \in G_i$.*

The function F is naturally extended to map interpreted systems of global states to Kripke models as follows: if $F(DS) = (W, R_1, \ldots, R_n)$ then $F(DS, \pi) = (W, R_1, \ldots, R_n, \pi)$.

Intuitively, the relations R_i represent an accessibility function to global states in which agent i is running according 'correct (or acceptable) operating circumstances'.

We illustrate this in Figure 1.

We make use of the construction above to give an interpretation to the formulas of a deontic language as follows. Given an interpreted deontic system $IDS = (DS, \pi)$, the interpretation of formulas of the language $\mathcal{L}$ is defined on the corresponding generated Kripke model $F(DS, \pi)$.

Definition 6 (Satisfaction on interpreted deontic systems of global states). *For any $\varphi \in \mathcal{L}$, $g \in DS$, and $IDS = (DS, \pi)$, satisfaction is defined by:*

$$IDS \models_g \varphi \text{ if } F(DS, \pi) \models_g \varphi,$$

where this is defined as:

$F(DS, \pi) \models_g$ **true**
$F(DS, \pi) \models_g p$ *if* $g \in \pi(p)$
$F(DS, \pi) \models_g \neg\varphi$ *if not* $F(DS, \pi) \models_g \varphi$
$F(DS, \pi) \models_g \mathcal{O}_i \, \varphi$ *if for all g' we have that $g \, R_i \, g'$ implies* $F(DS, \pi) \models_{g'} \varphi$.

In other words, the truth of formula $\mathcal{O}_i \, \varphi$ at a global state signifies the truth of formula φ in all the global states in which agent i is in a correct local state, i.e. in a green state.

Validity on deontic systems is defined similarly.

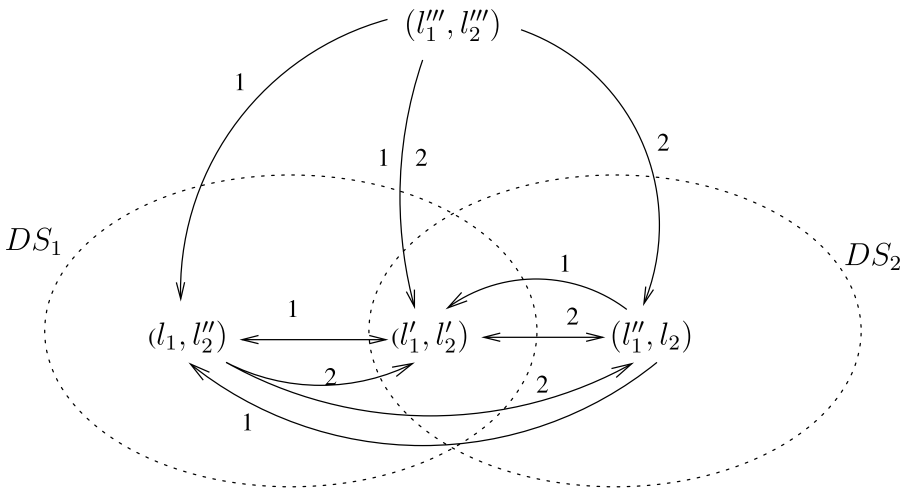


Fig. 1. An example of deontic system and its generated frame. In the example above the environment is not considered and the local states for the agents are composed as follows. Agent 1: $L_1 = \{l_1, l_1', l_1'', l_1'''\}, G_1 = \{l_1, l_1'\}$. Agent 2: $L_2 = \{l_2, l_2', l_2'', l_2'''\}, G_2 = \{l_2, l_2'\}$. $DS = \{(l_1', l_2'), (l_1, l_2''), (l_1'', l_2), (l_1''', l_2''')\}$. In the figure the sets DS_1, DS_2 represent the subsets of DS which present acceptable configurations respectively for agent 1, and 2. The labelled links indicate the relations R_1 and R_2 of the generated frame.

Definition 7 (Validity on deontic systems). *For any $\varphi \in \mathcal{L}$, and $IDS = (DS, \pi)$, validity on interpreted deontic systems of global states is defined by $IDS \models \varphi$ if $F(DS, \pi) \models \varphi$. For any $\varphi \in \mathcal{L}$, and $DS \in \mathcal{DS}$, validity on deontic systems of global states is defined by $DS \models \varphi$ if $F(DS) \models \varphi$.*

For any $\varphi \in \mathcal{L}$, we say that φ is valid *on the class $\mathcal{DS}$, and write $\mathcal{DS} \models \varphi$, if for every $DS \in \mathcal{DS}$ we have that $DS \models \varphi$.*

In the following we investigate the logical properties that deontic systems of global states inherit. From Definition 7 it follows that this analysis can be carried out on the class of the generated frames.

3 Axiomatisation of Deontic Interpreted Systems

In this section[2] we study deontic systems of global states from the axiomatic point of view. An immediate consideration comes from the following.

Lemma 1. *Given any DS, we have that $F(DS)$ is serial, transitive, and Euclidean.*

This observation leads immediately to the conclusion that the logic of deontic systems of global states must be at least as strong as KD45$_n$, which is to be expected. However, as

[2] Proofs of many of the results in this section are not included in this version. We refer the interested reader to [11] for details.

will be clearer in the following, it turns out that the logic determined by deontic systems of global states is in fact stronger than KD45$_n$. In order to see this, we need to introduce a few semantic structures.

3.1 Some Secondary Properties of Kripke Frames

In order to prove an axiomatisation for deontic systems of global states, we introduce some *secondary properties* of Kripke frames, by which we mean properties that hold on any sub-frame that can be reached from some point in the frame. (The term 'secondarily reflexive' is used in [1, p92].)

Lemma 2. *Let R be a binary relation on W. R is Euclidean iff R is universal on $R(w)$ for all $w \in W$.*

Definition 8 (Secondarily universal). *Let R be a binary relation on W. R is* secondarily universal *if*

(i) for all $w \in W$, R is universal on $R(w)$;
(ii) for all $w', w'' \in W$, $R(w') = R(w'')$.

A frame $F = (W, R_1, \ldots, R_n)$ is a secondarily universal frame if every relation R_i, $i \in A$, is secondarily universal.

It follows (by Lemma 2) that condition (i) of the definition is equivalent to the requirement that R is Euclidean. We have then that every secondarily universal relation is Euclidean.

We are now in a position to relate validity on the class of serial secondarily universal frames to validity on the class of serial, transitive and Euclidean frames. However, we are interested here in the multi-modal case, and for this we need a property of frames we call *i-j Euclidean*.

Definition 9 (i-j Euclidean frame). *A frame $F = (W, R_1, \ldots, R_n)$ is* i-j Euclidean *if for all $w, w', w'' \in W$, and for all $i, j \in A$, we have that $w\, R_i\, w', w\, R_j\, w''$ implies $w''\, R_i\, w'$.*

The class of i-j Euclidean frames collapses to 'standard' Euclidean frames for $i = j$.

There is a precise correspondence that can be drawn between i-j Euclidean frames and the following axiom:

$$\mathcal{P}_i\, p \rightarrow \mathcal{O}_j\, \mathcal{P}_i\, p \qquad \text{(for any } i, j \in A) \qquad 5^{i\text{-}j}$$

Lemma 3. *A frame F is i-j Euclidean if and only if $F \models 5^{i\text{-}j}$.*

Now we can relate validity on the class of (serial) secondarily universal frames to validity on the class of (serial) transitive, i-j Euclidean frames.

Lemma 4. *If a frame F is secondarily universal then it is also i-j Euclidean.*

Lemma 5. *Let $F = (W, R_1, \dots, R_n)$ be an i-j Euclidean frame. Then for every $i \in A$, and for all $w, w', w'' \in W$, if wR^*w' and wR_iw'' then $w'R_iw''$, where R^* is the reflexive transitive closure of $\cup_{i=1}^n R_i$.*

We now prove that the class of serial, transitive, i-j Euclidean frames and the class of serial, secondarily universal frames are semantically equivalent, that is, the same set of formulas φ is valid on both.

Theorem 1. *For all $\varphi \in \mathcal{L}$, φ is valid on the class of serial, secondarily universal frames if and only if φ is valid on the class of serial, transitive and i-j Euclidean frames.*

Theorem 2. *The logic $KD45_n^{i\text{-}j}$ is sound and complete with respect to*

- *serial, transitive and i-j Euclidean frames*
- *serial, secondarily universal frames.*

Proof. We show the first part; the second part follows immediately by Theorem 1. We show that the logic $\mathrm{KD45}_n^{i\text{-}j}$ is canonical, i.e. that the frame F_C of the canonical model M_C is serial, transitive, and i-j Euclidean. Since the logic in question is stronger than $\mathrm{KD45}_n$, from the literature we know that F_C is serial and transitive; we show it is i-j Euclidean. To do so, consider three maximal consistent sets w, w', w'', such that wR_iw', wR_jw''. It remains to show that $w''R_iw'$. By contradiction suppose this is not the case; then there must exist a formula $\alpha \in \mathcal{L}$ such that $\mathcal{O}_i\, \alpha \in w''$, and $\neg\alpha \in w'$. But then $\mathcal{P}_i \neg\alpha \in w$, and so $\mathcal{O}_j\, \mathcal{P}_i \neg\alpha \in w$, which in turn implies $\mathcal{P}_i \neg\alpha \in w''$, i.e. $\neg\, \mathcal{O}_i\, \alpha \in w''$, which is absurd, because it would make w'' inconsistent.

Before we can axiomatise deontic systems of global states we need to make clear the correspondence between deontic systems of global states and secondarily universal frames.

Theorem 3. *Any serial, secondarily universal frame is the p-morphic image of the frame generated by an appropriate deontic system of global states.*

Proof. Let $F = (W, R_1, \dots, R_n)$ be any serial secondarily universal frame; we define a deontic system DS as follows. Pick some $\overline{w} \in W$. For any relation $i \in A$, let $G_i = R_i(\overline{w})$. Since R_i is serial, this satisfies the requirement that G_i is not empty. Let $G_e = W$.

Consider now the deontic system DS defined as $DS = \{(w, w, \dots, w) \mid w \in W\}$. For simplicity we use the shortcut $(w, w, \dots, w) = [w]$. The frame generated by DS is defined (see Definition 5) by $F(DS) = F' = (W', R'_1, \dots, R'_n)$, where $W' = \{[w] \mid w \in W\}$, and for any $i \in A$ we have $R'_i = \{([w], [w']) \mid w' \in G_i\}$.

We show that F can be seen as the target of a p-morphism of domain F'. Define the function $p : W' \to W$ such that $p([w]) = w$.

- p is clearly surjective.
- For any $i \in A$, consider $[w]\, R'_i\, [w']$. By definition it must be $w' \in G_i$. So $\forall w'' \in W$ we have that $w''\, R_i\, w'$. But then in particular $w\, R_i\, w'$.

– Consider $w\, R_i\, p([w'])$. So, by construction, we have $p[w'] = w'$, and $w' \in G_i$. But then $[w]\, R'_i\, [w']$.

For the result presented in this paper, the notion of p-morphism is sufficient, but it can be noted that the function defined above is actually an isomorphism.

We can now prove the main result of this section.

Theorem 4. *The logic KD45$_n^{i\text{-}j}$ is sound and complete with respect to deontic systems of global states.*

Proof. The proof for soundness is straightforward and omitted here. For completeness, we prove the contrapositive. Suppose $\not\vdash \varphi$; then by Theorem 2, there exists a serial, secondarily universal model $M = (F, \pi)$ such that $M \not\models_w \varphi$, for some $w \in W$. By Theorem 3 there exists a deontic system DS such that $F(DS)$ is the domain of a p-morphism $p : F(DS) \to F$. But then by consideration on p-morphisms, since $F \not\models \varphi$, we have that $F(DS) \not\models \varphi$, so $DS \not\models \varphi$, so $\mathcal{DS} \not\models \varphi$, which is what we needed to show.

4 Discussion

4.1 The Logic KD45$_n^{i\text{-}j}$

In the previous section we showed that the logic KD45$_n^{i\text{-}j}$ provides a complete axiomatisation for deontic systems of global states. In the following we look at the individual axioms in a little more detail.

In light of much of the literature in this area the logic above should be seen as providing a *bird's eye view* of the properties of the MAS. Therefore validity of axiom K:

$$\mathcal{O}_i(p \to q) \to (\mathcal{O}_i\, p \to \mathcal{O}_i\, q) \qquad \text{K}$$

seems reasonable. Indeed, if agent i's functioning specification requires that whenever p is the case then q should also be the case, then, if according to the agent's functioning protocol p is the case, then q should also be the case according to that protocol.

Axiom D guarantees that individual specifications are consistent:

$$\mathcal{O}_i\, p \to \neg\, \mathcal{O}_i\, \neg p \qquad \text{D}$$

Another way of seeing the above is to note that in normal modal logics, axiom D is equivalent to $\neg\, \mathcal{O}_i$ **false**. Axiom D is sometimes called the characteristic deontic axiom: together with axiom K, axiom D is the basis for Standard Deontic Logic (SDL).

Moving to the next pair of axioms, if we give a bird's eye view reading of the $\mathcal{O}_i$ modality, then axiom 4

$$\mathcal{O}_i\, p \to \mathcal{O}_i\, \mathcal{O}_i\, p \qquad 4$$

and axiom 5

$$\mathcal{P}_i\, p \to \mathcal{O}_i\, \mathcal{P}_i\, p \qquad 5$$

are perhaps not as strong as a first reading might suggest.

Another way of reading axiom 4 is to note that it forbids the situation in which p is prescribed but it is allowed that p is not prescribed. This seems reasonable with respect to strong deontic notions such as the one we are modelling. For example consider the case of one agent running a program in which one of its variables is supposed to be 'guarded', say to a boolean value. It would then be unreasonable if the protocol were to specify that the variable has to be a boolean, but at the same time allowed it not to be prescribed that it be a boolean. It is worth pointing out that the underlying reason for the validity of axioms 4 and 5 in this context is that the criterion for what counts as a green state is *absolute*, that is to say, the set of green states for an agent is independent of the state in which it currently is. An alternative would be to introduce functions $g_i : L_i \to 2^{L_i}$ to identify green states; but that seems to have less appeal in the present context and we do not explore it further.

Lastly, axiom $5^{i\text{-}j}$ of the previous section, of which axiom 5 is a special case, also reflects the absolute nature of the specification of 'green'. It represents an interaction between the states of correctly functioning behaviour of pairs of agents.

$$\mathcal{P}_i\, p \to \mathcal{O}_j\, \mathcal{P}_i\, p \qquad 5^{i\text{-}j}$$

$5^{i\text{-}j}$ expresses the property that if a state of the system can happen under the correct behaviour of one agent i, then the protocol of any agent j must allow this eventuality in any correct state that it specifies for j. Again, this seems a reasonable assumption. Suppose that agent i can follow its functioning protocol and reach a state coded by p. Axiom $5^{i\text{-}j}$ stipulates that in this case agent j's protocol cannot prescribe as admissible any states in which agent i does not have the opportunity to move to a state coded by p. In other words, axiom $5^{i\text{-}j}$ asserts a sort of *independence* in the interplay between agents. Naturally, we do not have the very strong property that all specifications are mutually consistent: $\mathcal{O}_i\, p \to \neg\, \mathcal{O}_j\, \neg p$ is *not* valid. However, $5^{i\text{-}j}$ provides a weak kind of mutual consistency: agent j's protocol cannot forbid the possibility of p for agent i if this is granted by agent i's protocol.

It is instructive to note that the logic $\mathrm{KD45}_n^{i\text{-}j}$ contains also the following generalisation of axiom 4:

$$\mathcal{O}_i\, p \to \mathcal{O}_j\, \mathcal{O}_i\, p \qquad 4^{i\text{-}j}$$

This can be checked semantically, or derived as follows: $\mathcal{O}_i\, p \to \mathcal{O}_i\, \mathcal{O}_i\, p$ (by 5); $\mathcal{O}_i\, \mathcal{O}_i\, p \to \mathcal{P}_i\, \mathcal{O}_i\, p$ (by D); $\mathcal{P}_i\, \mathcal{O}_i\, p \to \mathcal{O}_j\, \mathcal{P}_i\, \mathcal{O}_i\, p$ (by $5^{i\text{-}j}$); $\mathcal{O}_j\, \mathcal{P}_i\, \mathcal{O}_i\, p \to \mathcal{O}_j\, \mathcal{O}_i\, p$ (by the rule RK and the dual of $5^{i\text{-}j}$).

It is now easy to check that the logic $\mathrm{KD45}_n^{i\text{-}j}$ contains all axioms in the following scheme:

$$X_i\, p \leftrightarrow Y_j\, X_i\, p$$

where X_i is any one of $\mathcal{O}_i, \mathcal{P}_i$ and Y_j is any one of $\mathcal{O}_j, \mathcal{P}_j$. There are thus only $2n$ distinct modalities in the logic $\mathrm{KD45}_n^{i\text{-}j}$.

4.2 A Notion of 'Global' Correctness

So far we have described and discussed the use of a green and red state semantics for interpreting the indexed deontic operator of correct behaviour. There are several possible ways to extend these notions to model the notion of *globally correct functioning behaviour* of the MAS. For example, it is straightforward to augment the framework with another modality $\mathcal{O}$ capturing global correctness, interpreted in terms of G, the set of green states for the system as a whole, as follows:

$$F(DS, \pi) \models_g \mathcal{O}\,\varphi \text{ if for all } g' \in G \text{ we have that } F(DS, \pi) \models_{g'} \varphi.$$

There are several possible definitions of G, depending on the notion of global correctness we wish to model:

1. $G = \{(l_e, l_1, \ldots, l_n) \mid l_e \in G_e\}$,
2. $G = \{(l_e, l_1, \ldots, l_n) \mid l_i \in G_i$ for some $i \in A\}$,
3. $G = \{(l_e, l_1, \ldots, l_n) \mid l_i \in G_i$ for all $i \in A\}$,

The first version corresponds to a notion of correct behaviour for the environment. This can be used to model system failures where these are associated with events such as communication breakdown, etc. In the second definition of G, a state of the system is regarded as correct whenever one or more of the agents in the system are in locally correct states; parts of the system might not be performing as intended but parts of it are. This can serve as a guarantee that the system is not completely crashed, as is the case, for example, in a system containing redundant components. It could also perhaps be used to model liveness. The third definition models correct states as states in which all the subcomponents are working correctly. This can be used to model a conservatory notion of correctness, useful when modelling safety critical systems.

Should the second definition from the list above be chosen as semantical model, the resulting axiomatisation would inherit the following interplay between globally and locally correct behaviours:

$$\mathcal{O}\,p \rightarrow \mathcal{O}_i\,p \qquad \text{for some } i \in A.$$

Should the third possibility be adopted, we would inherit the validity of:

$$\mathcal{O}\,p \rightarrow \mathcal{O}_i\,p \qquad \text{for all } i \in A.$$

It is also straightforward to generalise, to allow for the modelling of arbitrary groups of agents, and not just individual agents and the global system as a whole: $\mathcal{O}_X$ would represent correct functioning of any group of agents $X \subseteq A$, with $\mathcal{O}_X$ interpreted in various ways, in analogous fashion to the different notions of global correctness identified above. The indexed modality $\mathcal{O}_i$ is then the limiting case where X is a singleton $\{i\}$, and global correctness $\mathcal{O}$ is the limiting case where $X = A$. We leave detailed examination of these possibilities to future papers.

5 Further Work and Conclusions

In this paper we have explored the notion of deontic systems of global states and axiomatised the resulting semantics. This line of work lies within a growing body of research that deals with the issue of axiomatising Multi-Agent systems defined on computationally grounded semantics, in the sense of [17]. This approach was suggested in the influential work by Halpern and colleagues [3] in which the authors studied the axiomatic properties of different classes of interpreted systems. Recently, some works have tried to relate these issues to semantics with a finer level of detail [9]. Of particular importance in this line of research is the issue of proving completeness not just with respect to an abstract class of Kripke frames, but with respect to an automata like model of a distributed system, as done in this present paper. This line of analysis opens up interesting avenues with respect to performing model checking directly on these semantical classes in the spirit of [6].

Various technical issues seem to be worth exploring further. In particular the question of incorporating mentalistic attitudes on the semantics we used here seems interesting. We have already undertaken some analysis for the case of knowledge [12]. A similar analysis for the case of belief could perhaps lead to interesting remarks regarding an axiomatisation of the *s*incerity condition of the *inform* communicative act of the FIPA specification [4]. As the current specification stands, it states that a pre-condition for an *inform(s,r, p)* act is that sender s believes that p is the case. This strong condition has led to some criticism. Indeed, one can imagine various different scenarios in which deception is a technique that may be used by some agent. For example in a negotiation scenario it does not seem realistic to insist that the agents exchange the true limit price for the goods or service they are negotiating upon. In this context, it seems of some interest to explore the possibility of modifying the sincerity condition into $\mathcal{O}_s\, B_s p$. In this way, agent s would violate its own (perhaps certified by a third party) functioning protocol by sending an untruthful information, but it would still be practically possible for it to do so (and possibly face the consequences of this later on). The implications of this seem worth exploring; we do not carry out this exercise here.

Another line of work which we have recently started pursuing concerns the evaluation of the applicability of the present framework to modelling real scenarios, where one is typically required to reason about correct functioning behaviour of the agents and of the system as a whole. We briefly report only two cases here.

Agents communicating over a channel. Consider an operating systems scenario in which two agents communicate over a channel. Agent 1 is a monitor, agent 2 is a cron job. Let us assume that agent 2 occasionally sends information to agent 1 via the channel. Each agent may or may not be in a faulty state. We represent the agents' local states as $L_i = \{+, -\}, i = \{1, 2\}$, where the symbol $+$ represents a state of correct functioning behaviour, whereas $-$ stands for a faulty state; we capture the functioning behaviour of the communication channel in the same way. Clearly the set of states of the system consists of all the tuples $DS = \{(l_1, l_2, l_e)\}$ in which all the components vary on $\{+, -\}$. Consider now the atoms $P = \{p, q\}$ where p stands for "agent 1 is alive, i.e. listening", and q "agent 2 is sending messages". We would like to consider formulas like:

$$\mathcal{O}_1\, p \wedge \neg\, \mathcal{O}_2\, q \wedge \neg\, \mathcal{O}_2\, \neg q,$$

representing models in which agent 1 is supposed to be monitoring the channel (indeed we would model p as being true only in the global states where agent 1 is functioning correctly), whereas agent 2 may or may not be sending messages. Note that, when taking the bird's eye view, there is nothing problematic about the reading of formulas such as $\mathcal{O}_2\, p$ and $\mathcal{O}_1\, q$. Consider now a notion of global correctness corresponding to the environment, i.e. the communication channel in our scenario, working correctly. One may want to check whether or not particular models satisfy the formula:

$$\mathcal{O}(q \rightarrow p),$$

i.e., models in which whenever the communication channel is operational, it follows that all messages are successfully received. Note that this does not hold in the case of the full Cartesian product described here. An interesting question is what sort of additional enforcement mechanism it is possible to impose so that some wanted property, like the one above, is exhibited.

Financial liquidity. Consider the case of a financial institution in which the current accounts of customers are modelled as agents. Agents can perform different actions in the financial environment (arranging overdraft etc.) and perform actions affecting other agents (bank transfer etc.), so it is meaningful to take this perspective. Let us suppose that the agents' local states simply represent the balance of the account. We can introduce a notion of individual violation for the agents representing that the account must not be overdrawn, i.e. the set of red states for every agent is the one containing, say, all negative natural numbers. An agent being overdrawn clearly poses no risk to the general financial health of the institution, but the overall balance of the set of agents surely does. It is easy to see how to model these issues in terms of a global/local violations as described above by considering an appropriate set of propositional variables. But further, one would want to devise an enforcement mechanism that forces no global violation (i.e. all the agents collectively being in the black) while allowing for individuals to violate their individual constraints.

The examples above show that while the formalism seems adequate for representing local and global violations, it needs to be made stronger to represent more interesting concepts, such as regulation, and various kinds of enforcement/compliance mechanisms. Indeed from the syntactical/proof-theoretical point of view we have no more than a stronger version of Standard Deontic Logic, and therefore we are only likely to inherit its well known limitations, such as it inability to model contrary to duty structures [16]. We see this paper as a starting point. Our current plan is to investigate the extent to which generalisations of the present formalism to incorporate temporal evolution and action can be used to represent enforcement/compliance mechanisms.

Acknowledgements

This work was supported by the EU-funded FET project ALFEBIITE (IST-1999-10298).

References

1. B. Chellas. *Modal Logic*. Cambridge University Press, Cambridge, 1980.
2. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.
3. R. Fagin, J. Y. Halpern, and M. Y. Vardi. What can machines know? On the properties of knowledge in distributed systems. *Journal of the ACM*, 39(2):328–376, April 1992.
4. FIPA: Foundation for intelligent physical agents. http://www.fipa.org.

5. J. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990. A preliminary version appeared in *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, 1984.
6. J. Halpern and M. Vardi. *Model checking vs. theorem proving: a manifesto*, pages 151–176. Artificial Intelligence and Mathematical Theory of Computation. Academic Press, Inc, 1991.
7. G. E. Hughes and M. J. Cresswell. *A New Introduction to Modal Logic*. Routledge, New York, 1996.
8. S. A. Kripke. Semantic analysis of modal logic (abstract). *Journal of Symbolic Logic*, 24:323–324, 1959.
9. A. Lomuscio, R. van der Meyden, and M. Ryan. Knowledge in multi-agent systems: Initial configurations and broadcast. *ACM Transactions of Computational Logic*, 1(2), October 2000.
10. A. Lomuscio and M. Ryan. On the relation between interpreted systems and Kripke models. In M. Pagnucco, W. R. Wobcke, and C. Zhang, editors, *Agent and Multi-Agent Systems - Proceedings of the AI97 Workshop on the theoretical and practical foundations of intelligent agents and agent-oriented systems*, volume 1441 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, May 1998.
11. A. Lomuscio and M. Sergot. Investigations in grounded semantics for multi-agent systems specifications via deontic logic. Technical report, Imperial College, London, UK, 2000.
12. A. Lomuscio and M. Sergot. Extending interpreting systems with some deontic concepts. In J. van Benthem, editor, *Proceedings of TARK 2001*, pages 207—218, San Francisco, CA, July 2001. Morgan Kauffman.
13. J.-J. Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*, volume 41 of *Cambridge Tracts in Theoretical Computer Science*. Cambdridge University Press, 1995.
14. J.-J. Ch. Meyer and R. J. Wieringa. *Deontic Logic: A Concise Overview*, chapter 1, pages 3–16. Wiley Professional Computing Series. John Wiley and Sons, Chichester, UK, 1993.
15. S. Popkorn. *First Steps in Modal Logic*. Cambridge University Press: Cambridge, England, 1994.
16. H. Prakken and M. Sergot. Contrary-to-duty obligations. *Studia Logica*, 57(1):91–115, 1996.
17. M. Wooldridge. Computationally grounded theories of agency. In E. Durfee, editor, *Proceedings of ICMAS, International Conference of Multi-Agent Systems*. IEEE Press, 2000.
18. M. Wooldridge. *Reasoning about rational agents*. MIT Press, July 2000.

Agents and Roles: Refinement in Alternating-Time Temporal Logic

Mark Ryan[1] and Pierre-Yves Schobbens[2]

[1] School of Computer Science
University of Birmingham
Birmingham B15 2TT, UK
www.cs.bham.ac.uk/~mdr
M.D.Ryan@cs.bham.ac.uk
[2] Institut d'Informatique
Facultés Universitaires de Namur
Rue Grandgagnage 21
5000 Namur, Belgium
www.info.fundp.ac.be/~pys
pys@info.fundp.ac.be

Abstract. We present a notion of refinement between agent-oriented systems defined using alternating-time temporal logic (ATL). The refinement relation provides a framework for defining *roles* in a society of interacting agents, and formalising a relation of *conformance* between agents and roles. The refinement relation also allows us to construct abstractions in order to make verification more tractable.

1 Introduction

A distributed system may be seen as a collection of agents interacting together. This horizonal decomposition of a system should be integrated with a vertical decomposition, in which we refine abstract descriptions of agents into more concrete ones during the course of the system development. The abstract descriptions can be thought of as roles, which are fulfilled by the concrete agents.

Verifying that a set of agents fulfills its role must be done in the context of the other agents. This idea has lead to the assume/guarantee paradigm: the specification of an agent guarantees its behaviour will be correct, assuming that its environment (formed in part by other agents) is correct. Our goal is to integrate this idea with a framework for agent refinement.

To formalise these intuitions, we propose to use Alternating-time Temporal Logic [3] (ATL), which allows us to describe precisely the properties of the different agents involved in a system, and the *strategies* they have for achieving their goals. ATL is a branching temporal logic based on game theory. It contains the usual temporal operators (next, always, until) plus cooperation modalities $\langle\langle A\rangle\rangle$, where A is a set of players (also called agents). This modality quantifies over the set of behaviours. The formula $\langle\langle A\rangle\rangle\phi$ means that the agents in A have a collective strategy to enforce ϕ, whatever the choices of the other agents. ATL generalises CTL, and similarly ATL* generalises CTL*, and μ-ATL generalises the μ-calculus. These logics can be model-checked by generalising the techniques of CTL, often with the same complexity as their CTL counterpart.

We motivate refinement between agent-oriented systems. Our aim is to present a framework which provides

– A method for constructing systems by assembling components. The system is specified as a collection of roles, and the component agents are shown to conform to the roles.
– A method for verifying agents efficiently, by abstracting their environment (similar to [6], but generalising from LTL to ATL). An agent is verified under the guarantees made by the roles of its environment. This addresses the state explosion problem, since it is simpler to verify against the roles of the environment than against the agents implementing the roles.

We define a very general notion of refinement between alternating-time transition systems (ATS, the models of ATL), which allows splitting or coalescing sets of agents so that a role may be fulfilled by several agents, or more generally several roles fulfilled by several agents. As such, our refinement relation generalises that of [2]. As well as providing a framework for describing agents and the roles they conform to, the refinement relation can be used to construct abstractions that allow us to verify systems efficiently in the manner of Cadence SMV [6].

Our work could be extended to other logics which have been developed for modelling cooperation among agents, such as Wooldridge's [11,12], which builds upon that of Werner [10] and Singh [9]. We chose ATL because of its closeness to CTL, the existence of a linear-time model checking algorithm for it, and its associated model checker Mocha [1].

Outline. In section 2 we present a motivating example. Next, in section 3 we recall the basic concepts of ATL and ATL* and their semantics on ATSs. Section 4 defines refinement and shows some of its properties, and continues the development of the example. Finally in section 5 we discuss some directions for future work.

2 Refinement between Agents: Example

The sliding-window protocol (SWP) is a communications protocol designed to guarantee communication over an unreliable channel. The channel may loose, duplicate, and re-order messages. We assume, however, that the channel is not completely unreasonable; thus,

– Although it might loose some messages, it does not loose all of them. More precisely, if a given message is sent often enough, it will eventually get through.

– Although it might re-order messages, the degree to which it re-orders is bounded. The channel will not swap a given message with other messages more than N times.

The agents involved in the abstract specification of the SWP are depicted in Figure 1. Here is a very abstract account of the SWP. The message to be sent is

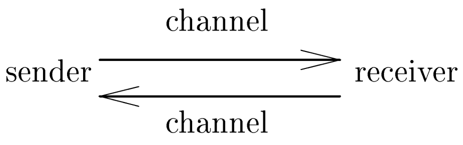


Fig. 1. Roles for the sliding-window protocol

split into packets, and the packets are passed in order to the sender. The sender's job is to send the packets to the receiver, over the unreliable channels. In order to cope with the deficiencies of the channels, the sender may send individual packets multiple times, until an acknowledgment is received from the receiver. Since the acknowledgments are also sent along an unreliable channel, they might need to be sent several times too. In order to cope with re-ordering, the sender labels messages with numbers consecutively chosen from the set $\{0, \ldots, M\}$, where $M > N$. When all the numbers are used up, the sender starts again from 0. The sender maintains an interval $[i, i + w]$, known as its window, of size $w < N$. Initially $i = 0$. The sender non-deterministically picks a packet whose sequence number is in the window, and sends it. It records any acknowledgments it receives; when it has received an acknowledgment for the ith message, it may advance its window by incrementing i. The receiver, whose job is to assemble the incoming message and send out acknowledgments, may be described in similar abstract terms.

This account is highly non-deterministic; it does not specify what order the sender should send the packets in its window, or what order the receiver should send the acknowledgments; this is inefficient. It allows the sender to resend messages which have already been acknowledged, which is wasteful. A more concrete specification of the SWP is given in [5], where the sender is split into three processes: the *trans*, which sends the packets initially and in order; the *ack-recv*, which records which messages have been acknowledged, and the *re-trans*, which re-transmits packets for which an acknowledgment has not been received before a given timeout. The reciever is also split into several components, as shown in Figure 2. The agents trans, ack-rcvr, and re-trans jointly play the role of sender, and accept and ack-send jointly play the role of receiver.

When proving properties of the *concrete* agents, we assume only the guarantees made by the *roles* of the other agents. This is a weaker assumption than the guarantees actually provided by the other agents, but it is leads to more efficient verification. Indeed, in a multi-layered example, while proving the prop-

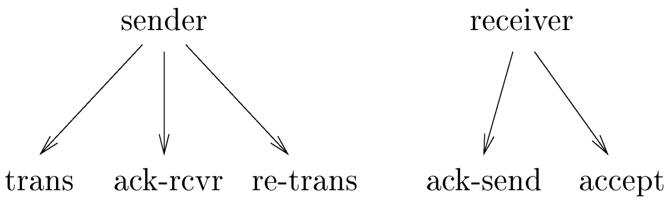


Fig. 2. Agents and Roles for the sliding-window protocol

erties of one agent we would assume the guarantees of the other agents in the most abstract layer possible.

3 Alternating-Time Temporal Logic

To make these intuitions precise, we formulate them in terms of Alternating-time Temporal Logic (ATL) [3]. ATL is based on CTL. Let us first recall a few facts about CTL. CTL [4] is a branching-time temporal logic in which we can express properties of reactive systems. For example, properties of cache-coherence protocols [7], telephone systems [8], and communication protocols have been expressed in CTL. One problem with CTL is that it does not distinguish between different sources of non-determinism. In a telephone system, for example, the different sources include individual users, the environment, and internal non-determinism in the telephone exchange. CTL provides the A quantifier to talk about all paths, and the E quantifier to assert the existence of a path. $\mathrm{A}\psi$ means that, no matter how the non-determinism is resolved, ψ will be true of the resulting path. $\mathrm{E}\psi$ asserts that, for at least one way of resolving the non-determinism, ψ will hold. But because CTL does not distinguish between different types of non-determinism, the A quantifier is often too strong, and the E quantifier too weak. For example, if we want to say about a telephone system that *user i can converse with user j*, CTL allows us to write the formulas $\mathrm{A}\Diamond \mathit{talking}(i,j)$ and $\mathrm{E}\Diamond \mathit{talking}(i,j)$. The first one says that in all paths, somewhere along the path there is a state in which i is talking to j, and is clearly much stronger than the intention. The second formula says that there is a path along which i is eventually talking j. This formula is weaker than the intention, because to obtain that path we may have to make choices on behalf of all the components of the system that behave non-deterministically. What we wanted to say is that users i and j can make their non-deterministic choices in such a way that, no matter how the other users or the system or the environment behaves, all the resulting paths will eventually have a state in which i is talking j. These subtle differences in expressing the properties we want to check can be captured accurately with ATL.

Alternating-time temporal logic (ATL) [3] generalises CTL by introducing *agents*, which represent different sources of non-determinism. In ATL the A and E path quantifiers are replaced by a path quantifier $\langle\!\langle A \rangle\!\rangle$, indexed by a subset A

of the set of agents. The formula $\langle\langle A\rangle\rangle\psi$ means that the agents in A can resolve their non-deterministic choices such that, no matter how the other agents resolve their choices, the resulting paths satisfy ψ. We can express the property that user i has the power, or capability, of talking to j by the ATL formula[1]

$$\langle\langle i\rangle\rangle\Diamond\texttt{talking(}i\texttt{,}j\texttt{)}\,.$$

We read $\langle\langle A\rangle\rangle\psi$ as saying that the agents in A can, by cooperating together, force the system to execute a path satisfying ψ. If A is the empty set of agents, $\langle\langle A\rangle\rangle\psi$ says that the system will execute a path satisfying ψ without the cooperation of any agents at all; in other words, $\langle\langle\emptyset\rangle\rangle\psi$ is equivalent to $\mathrm{A}\psi$ in CTL. Dually, $\langle\langle\Sigma\rangle\rangle\psi$ (where Σ is the entire set of agents) is a weak assertion, saying that if all the agents conspire together they may enforce ψ, which is equivalent to $\mathrm{E}\psi$ in CTL.

3.1 ATL and ATL*

Let P be a set of atomic propositions and Σ a set of agents. The syntax of ATL is given by

$$\phi ::= p \mid \top \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \langle\langle A\rangle\rangle[\phi_1 \;\mathrm{U}\; \phi_2] \mid \langle\langle A\rangle\rangle\Box\phi_1 \mid \langle\langle A\rangle\rangle\bigcirc\phi_1$$

where $p \in P$ and $A \subseteq \Sigma$. We use the usual abbreviations for $\rightarrow$, $\wedge$ in terms of $\neg$, $\vee$. The operator $\langle\langle\ \rangle\rangle$ is a path quantifier, and $\bigcirc$ (*next*), $\Box$ (*always*) and U (*until*) are temporal operators. The logic ATL is similar to the branching-time logic CTL, except that path quantifiers are parameterised by sets of agents. As in CTL, we write $\langle\langle A\rangle\rangle\Diamond\phi$ for $\langle\langle A\rangle\rangle[\top \;\mathrm{U}\; \phi]$, and we define the weak-until: $\langle\langle A\rangle\rangle[\phi \;\mathrm{W}\; \psi] = \neg[[A]][\neg\psi \;\mathrm{U}\; (\neg\phi \wedge \neg\psi)]$.

While the formula $\langle\langle A\rangle\rangle\psi$ means that the agents in A can cooperate to make ψ true (they can "enforce" ψ), the dual formula $[[A]]\psi$ means that the agents in A cannot cooperate to make ψ false (they cannot "avoid" ψ). The formulas $[[A]]\Diamond\phi$, $[[A]]\Box\phi$, and $[[A]]\bigcirc\phi$ stand for $\neg\langle\langle A\rangle\rangle\Box\neg\phi$, $\neg\langle\langle A\rangle\rangle\Diamond\neg\phi$, and $\neg\langle\langle A\rangle\rangle\bigcirc\neg\phi$.

The CTL path quantifiers A (all paths) and E (some path) can be recovered in ATL as $\langle\langle\emptyset\rangle\rangle$ and $\langle\langle\Sigma\rangle\rangle$. The logic ATL* generalises ATL in the same way that CTL* generalises CTL, namely by allowing path quantifiers and temporal operators to be nested arbitrarily.

For a subset $A \subseteq \Sigma$ of agents, the fragment $\langle\langle A\rangle\rangle$-ATL of ATL consists of ATL formulas whose only modality is $\langle\langle A\rangle\rangle$, and that does not occur within the scope of a negation. The $\langle\langle A\rangle\rangle$-ATL* fragment of ATL* is defined similarly.

3.2 Alternating Transitions Systems

Whereas the semantics of CTL is given in terms of transition systems, the semantics of ATL is given in terms of *alternating transition systems* (ATSs). An ATS S over a set of atomic propositions P and a set of agents Σ is composed of

[1] We write $\langle\langle i\rangle\rangle\psi$ instead of $\langle\langle\{i\}\rangle\rangle\psi$.

(Q, π, δ, I, o), where Q is a set of states and $\pi : Q \to 2^P$ maps each state to the set of propositions that are true in it, and

$$\delta : Q \times \Sigma \to 2^{2^Q}$$

is a transition function which maps a state and an agent to a non-empty set of *choices*, where each choice is a non-empty set of possible next states. If the system is in a state q, each agent a chooses a set $Q_a \in \delta(q, a)$; the system will move to a state which is in $\bigcap_{a \in \Sigma} Q_a$. We require that the system is non-blocking and that the agents together choose a unique next state; that is, for every q and every tuple $(Q_a)_{a \in \Sigma}$ of choices $Q_a \in \delta(q, a)$, we require that $\bigcap_{a \in \Sigma} Q_a$ is a singleton. Similarly, the initial state is specified by $I : \Sigma \to 2^{2^Q}$. I maps each agent to a set of choices. The agents together choose a single initial state: for each tuple $(Q_a)_{a \in \Sigma}$ of choices $Q_a \in I(a)$, we require that $\bigcap_{a \in \Sigma} Q_a$ is a singleton.

In the sequel, we will consider ATS of a particular form: the choices of the agents are expressed by choosing the value of their variables. This is also the choice made in the language Mocha [1]. We therefore stipulate a function $o : P \to \Sigma$ that, for each propositional variable, gives its owner. We define $P_a = \{p \in P \mid o(p) = a\}$ to designate the propositional variables belonging to agent a, and $P_A = \{p \in P \mid o(p) \in A\}$ to a group A. The choices of an agent thus determine the value of its variables, and let the other vary freely:

$$\forall a \in \Sigma, q \in Q, Q_a \in \delta(q, a), \forall X \subseteq P_{\neq a}, \exists q \in Q_a : \pi(q) \cap P_{\neq a} = X$$

where $P_{\neq a} = \{p \in P | o(p) \neq a\}$. A signature is defined as this function o, together with its domain P and codomain Σ.

For two states q and q', we say that q' is a *successor* of q if, for each $a \in \Sigma$, there exists $Q' \in \delta(q, a)$ such that $q' \in Q'$. We write $\delta(q)$ for the set of successors of q; thus,

$$\delta(q) = \bigcap_{a \in \Sigma} \bigcup_{Q \in \delta(q,a)} Q$$

A computation of S is an infinite sequence $\lambda = q_0, q_1, q_2 \ldots$ of states such that (for each i) q_{i+1} is a successor of q_i. We write $\lambda[0, i]$ for the finite prefix $q_0, q_1, q_2, \ldots, q_i$.

Often, we are interested in the cooperation of a subset $A \subseteq \Sigma$ of agents. Given A, we define $\delta(q, A) = \{\bigcap_{a \in A} Q_a \mid Q_a \in \delta(q, a)\}$. Intuitively, when the system is in state q, the agents in A can choose a set $T \in \delta(q, A)$ such that, no matter what the other agents do, the next state of the system is in T. Note that $\delta(q, \{a\})$ is just $\delta(q, a)$, and $\delta(q, \Sigma)$ is the set of singleton successors of q.

Example 1 ([3]). Consider a system with two agents "user" u and "telephone exchange" e. The user may lift the handset, represented as assigning value true to the boolean variable "offhook". The exchange may then send a tone, represented by assigning value true to the boolean variable "tone". Initially, both variables are false. Clearly, obtaining a tone requires collaboration of both agents.

We model this as an ATS $S = (Q, \pi, \delta, I)$ over the agents $\Sigma = \{u, e\}$ and propositions $P = \{\text{offhook,tone}\}$. Let $Q = \{00, 01, 10, 11\}$. 00 is the state in which both are false, 01 the state in which "offhook" is false and "tone" is true, etc. (thus, $\pi(00) = \emptyset$, $\pi(01) = \{tone\}$, etc.). The transition function δ and initial states I are as indicated in the figure.

$\delta(q, a)$	u	e
00	$\{\{00, 01\}, \{10, 11\}\}$	$\{\{00, 10\}\}$
10	$\{\{10, 11\}\}$	$\{\{00, 10\}, \{01, 11\}\}$
01	$\{\{00, 01\}, \{10, 11\}\}$	$\{\{01, 11\}\}$
11	$\{\{10, 11\}\}$	$\{\{01, 11\}\}$
I	$\{\{00, 01\}\}$	$\{\{00, 10\}\}$

Fig. 3. The transition function of the ATS.

3.3 Semantics

The semantics of ATL uses the notion of *strategy.* A *strategy* for an agent $a \in \Sigma$ is a mapping $f_a : Q^+ \to 2^Q$ such that $f_a(\lambda \cdot q) \in \delta(q, a)$ with $\lambda \in Q^*$. In other words, the strategy is a recipe for a to make its choices. Given a state q, a set A of agents, and a family $F_A = \{f_a \mid a \in A\}$ of strategies, the *outcomes* of F_A from q are the set $out(q, F_A)$ of all computations from q where agents in A follow their strategies, that is,

$$out(q_0, F_A) = \{\lambda = q_0, q_1, q_2, \cdots \mid \forall i,\ q_{i+1} \in \delta(q_i) \cap \big(\bigcap_{a \in A} f_a(\lambda[0, i])\big)\}.$$

If $A = \emptyset$, then $out(q, F_A)$ is the set of all computations, while if $A = \Sigma$ then it consists of precisely one computation.

The semantics of ATL* is as CTL*, with the addition of:

– $q \models \langle\langle A \rangle\rangle \psi$ if there exists a set F_A of strategies, one for each agent in A, such that for all computations $\lambda \in out(q, F_A)$ we have $\lambda \models \psi$.

Remark 1. To help understand the ideas of ATL, we state below some validities, and surprising non-validities.

1. If $A \subseteq B$, then $\langle\langle A \rangle\rangle \psi \to \langle\langle B \rangle\rangle \psi$, and $[[B]]\psi \to [[A]]\psi$. Intuitively, anything that A can enforce can also be enforced by a superset B; and if anything that B is powerless to prevent cannot be prevented by a subset of B.
2. In CTL*, A distributes over $\wedge$. But in general in ATL*, $\langle\langle A \rangle\rangle(\psi_1 \wedge \psi_2)$ only implies $(\langle\langle A \rangle\rangle \psi_1) \wedge (\langle\langle A \rangle\rangle \psi_2)$. The first formula asserts that agents A can enforce $\psi_1 \wedge \psi_2$, while the second is weaker, asserting that A has a way to enforce ψ_1 and another, possibly incompatible, way to enforce ψ_2. Similarly, $\langle\langle A \rangle\rangle(\psi_1 \vee \psi_2)$ and $\langle\langle A \rangle\rangle \psi_1 \vee \langle\langle A \rangle\rangle \psi_2$ are different (for $A \neq \Sigma$). The first one asserts that agents A can enforce a set of paths each of which satisfies $\psi_1 \vee \psi_2$, but which of the two is true along a particular path might be chosen

by other agents. This is weaker than the second formula, which asserts that A can guarantee that all paths satisfy ψ_1, or A can guarantee that all paths satisfy ψ_2.

4 Refinement of Agents

In this section, we define refinement between ATSs. As illustrated by the example of section 2, a refinement chooses a particular set of agents to refine, and may abstract the complement set. The formal definition of refinement is split into two parts. Signature refinement deals with the symbols in the vocabulary. Next, ATS refinement extends a signature refinement, and deals with behaviours.

Definition 1. *Let Σ and Σ' be disjoint sets of agents with variables P and P' respectively. A* signature refinement from $A \subseteq \Sigma$ to $A' \subseteq \Sigma'$ *is: a relation $f \subseteq \Sigma \times \Sigma'$ between agents in both societies, and a relation $g \subseteq P \times P'$, such that:*

- *f links agents in A to agents in A', and agents outside A to agents outside A'. To define this formally, we extend f to sets of agents in the usual way: $f(B) = \{a' \mid \exists a \in B f(a, a')\}$ and $f^{-1}(B') = \{a \mid \exists a' \in B' f(a, a')\}$. A set of agents $B \in \Sigma$* corresponds to *$B' \in \Sigma'$ iff $B' = f(B)$ and $f^{-1}(B') = B$. Note that some sets of agents will not have a correspondent. We require that A corresponds to A' and $\bar{A}$ to $\bar{A}'$ (where $\bar{A} = \Sigma \setminus A$ and $\bar{A}' = \Sigma' \setminus A'$).*
- *g respects ownership: $g(p, p')$ implies $f(o(p), o(p'))$. Additionally, the more detailed description of the component should at least include the variables required in its role, thus g is a function from $P_A = \{p \mid o(p) \in A\}$, and conversely, g^{-1} is a function on $P_{\bar{A}'}$.*

Example [continued from section 2]. $A = \{\text{sender}\} \subseteq \Sigma = \{\text{sender}, \text{receiver}\}$. $A' = \{\text{trans, ack-rcvr, re-trans}\} \subseteq \Sigma' = \{\text{trans, ack-rcvr, re-trans, receiver}\}$. This refines the sender. In this case, f is shown as the left three arrows of figure 2, together with the pair (receiver, receiver).

Signature refinements have the following properties:

Theorem 1. *1. The identity is a signature refinement;*
2. Signature refinements compose: if (f_1, g_1) is a signature refinement from $A_1 \subseteq \Sigma_1$ to $A_2 \subseteq \Sigma_2$, and (f_2, g_2) is a signature refinement from $A_2 \subseteq \Sigma_2$ to $A_3 \subseteq \Sigma_3$, then $(f_1 \circ f_2, g_1 \circ g_2)$ is a signature refinement from $A_1 \subseteq \Sigma_1$ to $A_3 \subseteq \Sigma_3$.

Given such a signature refinement, we can now consider how to translate formulae. Not all formulae can be translated: they may involve agents or variables that have no correspondent. The translation T along f, g of a formula is thus: $T(p) = g(p)$ assuming g is functional on p; $T(\langle\langle A \rangle\rangle \phi) = \langle\langle f(A) \rangle\rangle T(\phi)$ assuming $f(A)$ corresponds to A; and the other logical operators translate unchanged. If the assumptions above are not fulfilled, the formula cannot be translated.

A refinement between signatures can sometimes be extended to a refinement between ATS $S = (\Sigma, Q, \pi, \delta, I)$ and $S' = (\Sigma', Q', \pi', \delta', I')$. We then say that the agents A' are conformant to the role A.

Definition 2. *Let $S = (\Sigma, Q, \pi, \delta, I)$ and $S' = (\Sigma', Q', \pi', \delta', I')$ be ATSs. An* ATS refinement *r extending a signature refinement $s = (f, g)$ from $A \subseteq \Sigma$ to $A' \subseteq \Sigma'$ is a triple (H, C, D), where:*

1. *$H \subset Q \times Q'$ is a relation between states such that corresponding variables have the same value, formally: if $H(q, q'), g(v, v')$, then $v \in \pi(q)$ iff $v' \in \pi'(q')$;*
2. *$C : H \to \delta'(q', A') \to \delta(q, A)$: C gives for any pair (q, q') in the relation H and for any choice possible for the component A' in q', a corresponding choice for the role A in q;*
3. *conversely $D : H \to \delta(q, \bar{A}) \to \delta'(q', \bar{A}')$;*

such that $\forall (q, q') \in H, \forall T \in \delta(q, \bar{A}), \forall R' \in \delta'(q', A') : (T \cap C(q, q')(R')) \times (D(q, q')(T) \cap R') \subseteq H$.

H relates states in the abstract system with those of the refined system. The intuition for C and D is this: any choices A' makes in the refined system must correspond to choices A makes in the original system. C maps A''s choices to A's choices. The environment goes the other way; in S' it is more abstract, so D maps $\bar{A}$'s choices to $\bar{A}'$'s choices. The 'such that' part of the definition guarantees that, after these choices have been resolved, the two systems are again in H-related states. Note that since T and $C(q, q')(R)$ together define the choices of all agents, their intersection is a singleton, and similarly for $D(q, q')(T) \cap R$.

If the agents A of the abstract system S are viewed as a role, then this definition ensures that the agents A' of S' conform to that role. Note that again the notion of conformance goes in opposite directions for the system and for its environment.

4.1 Some Properties of the Refinement Relation

Our definition is similar to the definition of alternating refinement in [2]. However, our definition is more general than theirs, since it allows the agents and the variables to be different in the two systems. Moreover, our definition is stricter than theirs, in the sense that in the case that the agents and variables are the same in the two systems, our definition implies their one. This is formalised as follows:

Remark 2. If $r = (f, g, H, C, D)$ is an ATS refinement from $S = (\Sigma, Q, \pi, \delta, I)$ over P to $S' = (\Sigma', Q', \pi', \delta', I')$ over P' and $\Sigma = \Sigma'$, $P = P'$, and f, g are identity, then $S \leq_{\bar{A}} S'$ in the sense of [2]. The converse does not hold.

Proof. The definition of $S \leq_{\bar{A}} S'$ is that there exists $H \subseteq Q \times Q'$ such that $H(q, q')$ implies $\pi(q) = \pi'(q')$ and $\forall T \in \delta(q, \bar{A}) \, \exists T' \in \delta'(q', \bar{A}) \, \forall R' \in \delta'(q', A) \, \exists R \in \delta(q, A) \, (T \cap R) \times (T' \cap R') \subseteq H$. Suppose $(f, g, H, C, D$ is an ATS refinement from S to S'. To prove $S \leq_{\bar{A}} S'$, we set: $T' = D(q,q')(T)$ and $R = C(q,q')(R')$. As R may depend on T as well as R' in the definition of $S \leq_{\bar{A}} S'$, while $C(q,q')(R')$ depends on R' but not on T in ours, the converse does not hold.

This difference between our definition and that of [2] has important consequences. Our definition has properties which their one does not have (such as Theorem 4 below), and the converse is also true. We discuss this further at the end of this section.

Now ATS refinements have the properties:

Theorem 2. *1. The identity is an ATS refinement;*

2. *ATS refinements compose vertically: if $r_1 = (f_1, g_1, H_1, C_1, D_1)$ is a ATS $S_1 \to S_2$ refinement from $A_1 \subseteq \Sigma_1$ to $A_2 \subseteq \Sigma_2$, and $r_2 = (f_2, g_2, H_2, C_2, D_2)$ is a $S_2 \to S_3$ refinement from $A_2 \subseteq \Sigma_2$ to $A_3 \subseteq \Sigma_3$, then there is a $S_1 \to S_3$ refinement r_3 from A_1 to A_3, which may also be written $r_1; r_2$, given by $(f_1 \circ f_2, g_1 \circ g_2, H_1 \circ H_2, C_1 \circ C_2, D_1 \circ D_2)$, where $\circ$ is the relation composition: $H_1 \circ H_2 = \{(q_1, q_3) \mid \exists q_2 (q_1, q_2) \in H_1 \wedge (q_2, q_3) \in H_2\}$.*
3. *If $r = (f, g, H, C, D)$ is an ATS $S_1 \to S_2$ refinement from $A_1 \subseteq \Sigma_1$ to $A_2 \subseteq \Sigma_2$, then $r^{-1} = (f^{-1}, g^{-1}, H^{-1}, C', D')$ is an $S_2 \to S_1$ refinement from $\bar{A_2}$ to $\bar{A_1}$, where C', D' are defined by: $C'(q_2, q_1) = D(q_1, q_2)$ and $D'(q_2, q_1) = C(q_1, q_2)$. Note that $S \leq_{\bar{A}} S'$ does not imply $S' \leq_A S$.*

Theorem 3. *If there is an ATS refinement from $A \subseteq \Sigma$ to $A' \subseteq \Sigma'$, then any translatable $\langle\langle A' \rangle\rangle, [[\bar{A}']]$-ATL formula valid on the ATS S' is also valid on S when translated; conversely, any translatable $\langle\langle \bar{A} \rangle\rangle, [[A]]$-ATL formula valid on S is also valid on S' when translated.*

Proof. Use remark 2 and theorem 6 of [2].

4.2 Horizontal Composition of Refinements

We use the refinement from $A \subseteq \Sigma$ to $A' \subseteq \Sigma'$ when we want to prove that the agents in A' satisfy the role A. The agents in A' are more *concrete* than those in A. As already noted, the definition of refinement means that the agents in $\bar{A}'$ may be more *abstract* than those in $\bar{A}$.

Now suppose we have two refinements, $r_1 = (f_1, g_1, H_1, C_1, D_1)$ and $r_2 = (f_2, g_2, H_2, C_2, D_2)$, such that r_1 is an $S \to S_1$ refinement from $A \subseteq \Sigma$ to $A_1 \subseteq \Sigma_1$, and a $S \to S_2$ refinement from $B \subseteq \Sigma$ to $B_2 \subseteq \Sigma_2$. Refinement r_1 refines the agents in A, and r_2 refines those in B. We would like to be able to put these refinements together, to form a refinement on $A \cup B$. This would allow us to implement the roles A and B independently, and verify the implementations independently. Putting the refinements together means that we obtain a correct refinement of $A \cup B$ with no additional verification. Recall that r_1 refines A, but may abstract B; and r_2, which refines B, may abstract A. Therefore, the composition of r_1 and r_2, noted $r_1 \| r_2$ in Figure 4, has to combine the refinement of A by r_1 and the refinement of B by r_2. We stipulate that $A \cap B$ is empty, in order to avoid the possibility that these refinements are incompatible on their common part.

Example [continued]. Let S be the abstract description of the sliding-window protocol (SWP), having agents $\Sigma = \{$sender, receiver, channel1, channel2$\}$. We decide to implement the sender and the receiver separately (Figure 5). The set $A = \{$sender$\}$ is implemented as $A_1 = \{$trans, ack-rcvr, re-trans$\}$. In this

$$\begin{tikzcd}
 & & & S & & & \\
 & & A,\ \bar{A} \arrow[dl, shift right, "r_1"'] & A{\cup}B,\ \bar{A}{\cap}\bar{B} \arrow[dd, shift right] \arrow[dd, dash, shift left, "r_1\|r_2"] & B,\ \bar{B} \arrow[dr, shift left] & & \\
S_1\ \Sigma_1 & A_1,\ \bar{A}_1 \arrow[ur, shift right] \arrow[dr, dash, shift right] \arrow[dr, shift left, "r_1'"'] & & & & B_2,\ \bar{B}_2 \arrow[ul, shift left, "r_2"'] \arrow[dl, dash, shift left] \arrow[dl, shift right, "r_2'"] & S_2\ \Sigma_2 \\
 & & & A_1{\cup}B_2,\ \bar{A}{\cap}\bar{B} & & & \\
 & & A_1,\ B_2{\cup}(\bar{A}{\cap}\bar{B}) & & B_2,\ A_1{\cup}(\bar{A}{\cap}\bar{B}) & & \\
 & & & S' & & & \\
 & & & \Sigma' = A_1{\cup}B_2{\cup}(\bar{A}{\cap}\bar{B}) & & &
\end{tikzcd}$$

Fig. 4. Horizontal composition of refinements.

example, $\bar{A}_1 = \bar{A} = \{$channel1, channel2, receiver$\}$. We check that this implementation, S_1, is a refinement of S from A to A_1.

The receiver is also implemented. $B_2 = \{$receiver$\}$ is refined into $B_2' = \{$accept, ack-send$\}$.

In the general framework of Figure 4 and Theorem 4 below, $\bar{A}_1$ is allowed to be an abstraction of $\bar{A}$. This is useful for model checing, because S_1 will be more abstract and so checking the refinement will be computationally cheaper.

Theorem 4. *ATS refinements compose horizontally: if $r_1 = (f_1, g_1, H_1, C_1, D_1)$ is an $S \to S_1$ refinement from $A \subseteq \Sigma$ to $A_1 \subseteq \Sigma_1$, and $r_2 = (f_2, g_2, H_2, C_2, D_2)$ is an $S \to S_2$ refinement from $B \subseteq \Sigma$ to $B_2 \subseteq \Sigma_2$ such that $A \cap B = \emptyset$, then we can build r, also noted $r_1\|r_2$, an $S \to S'$ refinement from $A \cup B \subseteq \Sigma$ to $A_1 \cup B_2 \subseteq \Sigma'$, such that there are refinements:*

1. *r_1', an $S_1 \to S'$ refinement from $\bar{A}_1 \subseteq \Sigma_1$ to $B_2 \cup (\bar{A} \cap \bar{B}) \subseteq \Sigma'$, and*
2. *r_2', an $S_2 \to S'$ refinement from $\bar{B}_2 \subseteq \Sigma_2$ to $A_1 \cup (\bar{A} \cap \bar{B}) \subseteq \Sigma'$,*

and any other such $S \to S'$ refinement r' from $A \cup B$ to C can be decomposed through r: there is r'' such that $r' = r; r''$. The situation is shown in figure 4.

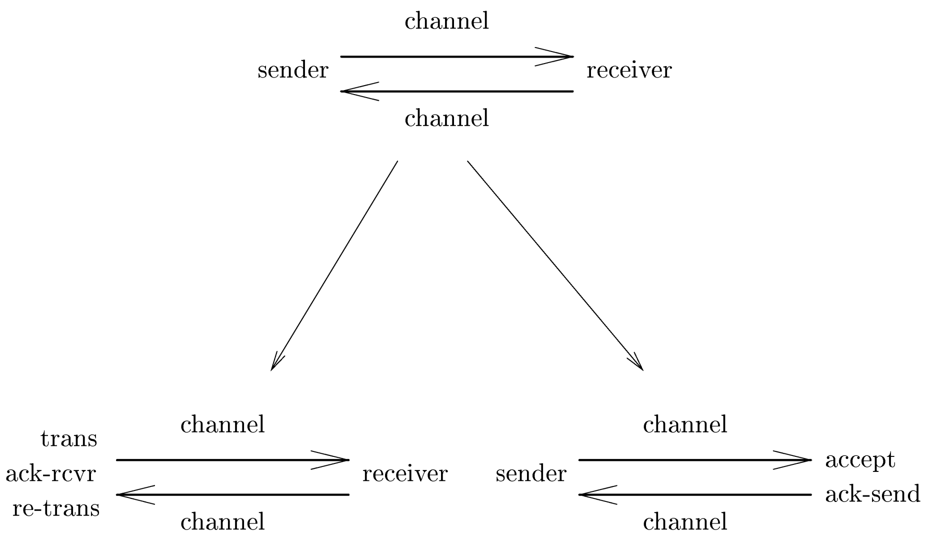


Fig. 5. Independently implementing the sender and the receiver.

Example [continued]. This shows that the two implementations can be put together, and that the result refines the original specification. Moreover, we do not have to check this (which would be computationally expensive, because of the size of S'); we simply check the refinements r_1 and r_2, and the theorem guarantees the refinement $r_1 \| r_2$.

Proof. We construct $S' = (\Sigma', Q', \pi', \delta', I')$ and the $S \rightarrow S'$ refinement $r = (f, g, H, C, D)$:

- $\Sigma' = A_1 \cup B_2 \cup (\bar{A} \cap \bar{B})$
- $Q' = \{(q, q_1, q_2) \mid H_1(q_1, q) \wedge H_2(q_2, q)\}$
- $v \in \pi'(q, q_1, q_2)$ if $o(v) \in A_1$ and $v \in \pi_1(q_1)$; or $o(v) \in B_2$ and $v \in \pi_2(q_2)$; or $o(v) \in \bar{A} \cap \bar{B}$ and $v \in \pi(q)$.
- $\delta'(a, (q, q_1, q_2)) =$
$$\begin{cases} \{C_1(q,q_1)(T_1) \times T_1 \times D_2(q,q_2)(C_1(q,q_1)(T_1)) \cap Q' \mid T_1 \in \delta_1(a,q_1)\} & \text{if } a \in A_1 \\ \{C_2(q,q_2)(T_2) \times D_1(q,q_1)(C_2(q,q_2)(T_2) \times T_2 \cap Q' \mid T_2 \in \delta_2(a,q_2)\} & \text{if } a \in B_2 \\ \{T \times D_1(q,q_1)(T) \times D_2(q,q_2)(T) \cap Q' \mid T \in \delta(q,a)\} & \text{if } a \in \bar{A} \cap \bar{B} \end{cases}$$
- $f(a, a')$ if $a \in A$ and $f_1(a, a')$; or $a \in B$ and $f_2(a, a')$; or $a \in \bar{A} \cap \bar{B}$ and $a = a'$.
- $g(v, v')$ if $o(v) \in A$ and $g_1(v, v')$; or $o(v) \in B$ and $g_2(v, v')$; or $o(v) \in \bar{A} \cap \bar{B}$ and $v = v'$.
- $H(q', (q, q_1, q_2)) \Leftrightarrow q = q'$
- $C(q, (q, q_1, q_2))(T \times T_1 \times T_2) = T$

– $D(q,(q,q_1,q_2))(T) = T \times D_1(q,q_1)(T) \times D_2(q,q_2)(T)$

It is straightforward to prove that S' is an ATS (the main part of which is to show that $\delta'(\Sigma',q)$ is a singleton). We prove that $r : S \to S'$ is a refinement:

1. f links $A \cup B \subseteq \Sigma$ to $C' = A_1 \cup B_2 \subseteq \Sigma'$.
2. g respects ownership: if $g(v,v')$ then, for instance, $o(v) \in A$ and $g_1(v,v')$; the second conjunct implies $f_1(o(v),o(v'))$, which together with $o(v) \in A$ implies $f(o(v),o(v'))$
3. g is functional on $P_{A\cup B}$, since g_1 is functional on P_A and g_2 functional on P_B.
4. We show $H(q',(q,q_1,q_2))$ and $g(v,v')$ imply $v \in \pi(q')$ iff $v' \in \pi'(q,q_1,q_2)$. Since $H(q',(q,q_1,q_2))$, $q = q'$. Now we distinguish between the usual three cases: $o(v)$ may be in A, B, or $\bar{A}\cap\bar{B}$. For example, if $o(v) \in A$ then $g(v,v')$ implies $g_1(v,v')$, therefore $o'(v') \in A_1$. Since $(q,q_1,q_2) \in Q'$, $H_1(q,q_1)$. Therefore, $v \in \pi(q')$ iff $v' \in \pi_1(q_1)$. The other cases are similar.
5. Finally, we need to show that if $H(q,(q,q_1,q_2))$, $T \in \delta(q,A\cup B)$, $R \in \delta'((q,q_1,q_2),\bar{A}\cap\bar{B})$, then $(T\cap C(q,(q,q_1,q_2))(R))\times(D(q,(q,q_1,q_2))(T)\cap R) \subseteq H$. Note that $T = \bigcap_{a\in A\cup B} Q_a$ for some choices $Q_a \in \delta(q,a)$, $a \in A\cup B$, and $R = \bigcap_{a\in\bar{A}\cap\bar{B}}(Q_a \times D_1(q,q_1)(Q_a) \times D_2(q,q_2)(Q_a)) \cap Q'$, for again some choices $Q_a \in \delta(q,a)$, $a \in \bar{A}\cap\bar{B}$. Let us fix these choices of Q_a, $a \in \Sigma'$. Suppose $(s,(s',s_1,s_2) \in (T \cap C(q,(q,q_1,q_2))(R)) \times (D(q,(q,q_1,q_2))(T) \cap R)$. Then $\{s\} = \bigcap_{a\in\Sigma'} Q_a$ and $\{(s',s_1,s_2)\} = \bigcap_{a\in\Sigma'}(Q_a \times D_1(q,q_1)(Q_a) \times D_2(q,q_2)(Q_a)) \cap Q'$. Therefore, $s = s'$.

The refinement $r_1' : S_1 \to S'$ is given by

– $f_1' \subseteq \Sigma' \times \Sigma_1$ given by: $f_1'(a',a_1)$ iff $a_1 = a' \in A_1$; or $a_1 \notin A_1$ and $a' \in B_2$ and $(a_1,a') \in f_2 \circ f_1$; or $a_1 \notin A_1$ and $a' \in \bar{A}\cap\bar{B}$ and $(a_1,a') \in f_1$.
– $H_1'(q_1',(q,q_1,q_2)) \Leftrightarrow q_1 = q_1'$
– $C_1'(q_1,(q,q_1,q_2))(T_1) = C_1(q_1,q)(T_1) \times T_1 \times D_2(q,q_2)(C_1(q_1,q)(T_1))$
– $D_1'(q_1,(q,q_1,q_2))(T \times T_1 \times T_2) = T_1$.

The refinement $r_2' : S' \to S_2$ is defined similarly, and we must again check that r_1 and r_2 are a refinement (omitted).

Finally, we show that any other such $S \to S'$ refinement r' from $A \cup B$ to C can be decomposed through r: there is r'' such that $r' = r; r''$: . . .

This property is not enjoyed by the refinement relation of [2], as the following example shows. Let $S = (\Sigma,Q,\pi,\delta,I)$ over P be given by: $\Sigma = \{a,b\}$, $Q = \{00,01,10,11\}$, $P = \{x,y\}$, $\pi(01) = \{y\}$, etc, and $\delta(a,q) = \{\{00,01\},\{10,11\}\}$ and $\delta(b,q) = \{\{00,10\},\{01,11\}\}$ (note: they are independent of q). Let S_1 be given by: $\Sigma_1 = \Sigma$, $Q_1 = Q$, and $\delta_1(a,q) = \{\{00,11\}\}$, $\delta_1(b,q) = \{\{00,10\},\{01,11\}\}$. Let S_2 be given by: $\Sigma_2 = \Sigma$, $Q_2 = Q$, and $\delta_2(a,q) = \{\{00,01\},\{10,11\}\}$, $\delta_2(b,q) = \{\{01,10\}\}$.

Note that $S_1 \leq_b S$ and $S_2 \leq_a S$. However it is not possible to construct an S' incorporating the choices of both S_1 and S_2, because S_1 insists that $x = y$ while S_2 insists that $x \neq y$.

There is no refinement from S to S_1, or from S to S_2, and therefore this counter-example does not apply to our definition.

5 Conclusions

Our refinement framework allows the system developer to implement different agents of a system independently. The results can be put together automatically to form an implementation of the whole. This guaranteed interoperability: in the example, any implementation of the sender is compatible with any implementation of the receiver.

Showing that the roles A and B may be refined independently is an important step in building a refinement calculus suitable for making abstractions to address the state explosion problem in ATL model checking. We intend to continue this work by efficiently computing refinements and developing examples.

A counterexample shows that our result about compositional refinements does not work for the refinement relation of [2], suggesting that our definition is more natural. However, their definition precisely characterises ATL formulas, as they show.

Acknowledgments

We are grateful to an anonymous referee for pointing out relevant literature.

References

1. R. Alur, H. Anand, R. Grosu, F. Ivancic, M. Kang, M. McDougall, B.-Y. Wang, L. de Alfaro, T. Henzinger, B. Horowitz, R. Majumdar, F. Mang, C. Meyer, M. Minea, S. Qadeer, S. Rajamani, and J.-F. Raskin. *Mocha User Manual.* University of California, Berkeley. `www.eecs.berkeley.edu/~mocha`.
2. R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In D. Sangiorgi and R. de Simone, editors, *CONCUR 98: Concurrency Theory*, Lecture Notes in Computer Science 1466, pages 163–178. Springer-Verlag, 1998.
3. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 100–109. IEEE Computer Society Press, 1997.
4. E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In D. Kozen, editor, *Logic of Programs Workshop*, number 131 in LNCS. Springer Verlag, 1981.
5. G. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1990.
6. K. McMillan. The SMV language. Available from www-cad.eecs.berkeley.edu/-~kenmcmil, June 1998.
7. K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.
8. M. C. Plath and M. D. Ryan. Feature integration using a feature construct. *Science of Computer Programming*, 2001. In print.
9. M. P. Singh. Group ability and structure. In Y. Demazeau and J.-P. Müller, editors, *Decentralised AI 2*, pages 127–146. Elsevier, 1991.

10. E. Werner. What can agents do together: A semantics of co-operative ability. In *Proc. ECAI 90*, pages 694–701, 1990.
11. M. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, UMIST, Manchester, 1992.
12. M. Wooldridge and M. Fisher. A first-order branching time logic of multi-agent systems. In *Proceedings of the Tenth European Conference on AI (ECAI-92)*, Vienna, Austria, 1992.

The Computational Complexity of Agent Verification

Michael Wooldridge and Paul E. Dunne

Department of Computer Science
University of Liverpool
Liverpool L69 7ZF
United Kingdom
{M.J.Wooldridge, P.E.Dunne}@csc.liv.ac.uk

Abstract. In this paper, we investigate the computational complexity of the *agent verification* problem. Informally, this problem can be understood as follows. Given representations of an agent, an environment, and a task we wish the agent to carry out in this environment, can the agent be guaranteed to carry out the task successfully? Following a formal definition of agents, environments, and tasks, we establish two results, which relate the computational complexity of the agent verification problem to the complexity of the task specification (how hard it is to decide whether or not an agent has succeeded). We first show that for tasks with specifications that are in Σ_u^p, the corresponding agent verification problem is Π_{u+1}^p-complete; we then show that for PSPACE-complete task specifications, the corresponding verification problem is no worse — it is also PSPACE-complete. Some variations of these problems are investigated. We then use these results to analyse the computational complexity of various common kinds of tasks, including achievement and maintenance tasks, and tasks that are specified as arbitrary Boolean combinations of achievement and maintenance tasks.

1 Introduction

We share with many other researchers the long-term goal of building agents that will be able to act autonomously in dynamic, unpredictable, uncertain environments in order to carry out complex tasks on our behalf. Central to the deployment of such agents is the problem of *task specification*: we want to be able to describe to an agent what task it is to carry out on our behalf *without* telling the agent *how* it is to do the task.

Associated with the issue of task specification are several fundamental computational problems. The first of these is the *agent design* problem: Given (specifications of) a task and an environment in which this task is to be carried out, is it possible to design an agent that will be guaranteed to successfully accomplish the task in the environment? The complexity of this problem was investigated in [10,11] for a range of different task types and representation schemes. Two main types of task were investigated in this work: *achievement tasks* and *maintenance tasks*. Crudely, an achievement task is specified by a set of "goal" states;

an agent is simply required to bring about one of these states in order to have succeeded. Maintenance tasks are also specified by a set of states, (referred to as "bad" states), but this time an agent is regarded as having succeeded if it *avoids* these states. The complexity of the agent design problem for both achievement and maintenance tasks was shown to be PSPACE-complete when reasonable assumptions were made about the representation of the task and the environment [10], although less tractable (more verbose) representations reduced the complexity of the problem, rendering it NL-complete in the best case considered. In [11], these results were shown to generalise to seemingly richer types of task specifications: *combined* task specifications, (in which tasks are specified as a pair consisting of an achievement and maintenance task — the agent is required to bring about the goal while maintaining the invariant), and *disjunctive* task specifications, (in which a task is specified as a set of pairs of achievement and corresponding maintenance tasks: on any given run, the agent is required to satisfy one of the goals while maintaining the corresponding invariant).

In this paper, we consider the closely related problem of *agent verification*. Informally, the agent verification problem is as follows. We are presented with (representations of) an environment, a specification of a task to be carried out in this environment, and an agent; we are asked whether or not the agent can be guaranteed to carry out the task successfully in the environment. When considering this problem, we view task specifications as predicates Ψ over execution histories or runs. In this paper, we examine and characterise the complexity of the agent verification problem for two general classes of task specification predicate. In the first case, we examine the complexity of the problem for task specifications Ψ that are (strictly) in Σ_u^p for some $u \in I\!N$. Notice that this includes perhaps the two most important complexity classes: P (i.e., Σ_0^p) and NP (i.e., Σ_1^p). We prove that if Ψ is in Σ_u^p then the corresponding agent verification problem is Π_{u+1}^p-complete. The second general result shows that if the task specification Ψ is PSPACE-complete, then the corresponding verification problem is no worse — it is also PSPACE-complete.

We then use these results to analyse a number of different task specification frameworks. In particular, we analyse the complexity of verification for achievement and maintenance tasks, and the complexity of a more general framework in which tasks are specified as arbitrary Boolean combinations of achievement and maintenance tasks.

We begin in the following section by setting up an abstract model of agents and environments, which we use to formally define the decision problems under study. We then informally motivate and introduce the various agent verification problems we study, and prove our main results. We discuss related work in section 5, and present some conclusions in section 6. Throughout the paper, we presuppose some familiarity with complexity theory [7].

2 Agents, Environments, and Tasks

In this section, we present an abstract formal model of agents, the environments they occupy, and the tasks that they carry out. (Please note that because of space restrictions, we simplified the presentation in this section by omitting some self-evident technical details.)

Environments

The systems of interest to us consist of an agent situated in some particular environment. The agent interacts with the environment by performing actions upon it, and the environment responds to these actions with changes in state. It is assumed that the environment may be in any of a finite set $E = \{e, e', \ldots\}$ of instantaneous states. Agents are assumed to have a repertoire of possible actions available to them, which transform the state of the environment. Let $Ac = \{\alpha, \alpha', \ldots\}$ be the (finite) set of actions.

The basic model of agents interacting with their environments is as follows. The environment starts in some state, and the agent begins by choosing an action to perform on that state. As a result of this action, the environment can respond with a number of possible states. However, only one state will *actually* result — though of course, the agent does not know in advance which it will be. On the basis of this second state, the agent again chooses an action to perform. The environment responds with one of a set of possible states, the agent then chooses another action, and so on.

A *run*, r, of an agent in an environment is thus a sequence of interleaved environment states and actions:

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{u-1}} e_u$$

Let $\mathcal{R}$ be the set of all such possible runs (over some Ac, E), let $\mathcal{R}^{Ac}$ be the subset of $\mathcal{R}$ that end with an action, and let $\mathcal{R}^E$ be the subset of $\mathcal{R}$ that end with a state. We use $r, r', \ldots$ to stand for members of $\mathcal{R}$.

In order to represent the effect that an agent's actions have on an environment, we introduce a *state transformer* function:

$$\tau : \mathcal{R}^{Ac} \to 2^E.$$

Thus a state transformer function is a partial function that maps a run (assumed to end with an allowable action of the agent) to a set of possible environment states. There are two important points to note about this definition. First, environments are allowed to be *history dependent* (or non-Markovian). In other words, the next state of an environment is not solely determined by the action performed by the agent and the current state of the environment. The previous actions performed by the agent, and the previous environment states also play a part in determining the current state. Second, note that this definition allows for non-determinism in the environment. There is thus *uncertainty* about the result of performing an action in some state.

If $\tau(r) = \emptyset$, (where r is assumed to end with an action), then there are no possible successor states to r. In this case, we say that the run is *terminated.* Similarly, if an agent has no allowable actions, then we say a run has terminated. One important assumption we make is that every run is guaranteed to terminate with length at most $|Ac \times E|$. This assumption may appear restrictive, and so some justification is necessary. The most important point is that while the restriction is of *theoretical* importance (it limits the generality of our results), it is not likely to be of *practical* importance, for the following reason. Consider an environment whose state is determined by k Boolean values; clearly, this environment can be in 2^k states. In practice, we will *never* have to consider runs that go on for 2^k time steps, for an environment defined by k Boolean attributes. So the restriction on run length, while being a theoretical limit to our work, is reasonable if we are concerned with tasks of practical, everyday interest.

Before proceeding, we need to make clear a couple of assumptions about the way that transformer functions are *represented.* To understand what is meant by this, consider that the input to the decision problems we study will include some sort of representation of the behaviour of the environment, and more specifically, the environment's state transformer function τ. We assume that the state transformer function τ of an environment is described as a two-tape Turing machine, with the input (a run and an action) written on one tape; the output (the set of possible resultant states) is written on the other tape. It is assumed that to compute the resultant states, the Turing machine requires a number of steps that is at most polynomial in the length of the input.

Formally, an environment *Env* is a quad $Env = \langle E, Ac, \tau, e_0 \rangle$ where E is a set of environment states, Ac is a set of actions, τ is a state transformer function, and $e_0 \in E$ is the initial state of the environment.

Agents

We model agents simply as functions that map runs (assumed to end with an environment state) to actions (cf. [9, pp580–581]):

$$Ag : \mathcal{R}^E \to Ac$$

Notice that while environments are implicitly non-deterministic, agents are assumed to be deterministic.

We say a *system* is a pair containing an agent and an environment. Any system will have associated with it a set of possible *runs*: formally, a sequence $(e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots, e_k)$ represents a run of an agent Ag in environment $Env = \langle E, Ac, \tau, e_0 \rangle$ iff

1. e_0 is the initial state of *Env* and $\alpha_0 = Ag(e_0)$; and
2. for $u > 0$

$$\begin{aligned} e_u &\in \tau((e_0, \alpha_0, \ldots, \alpha_{u-1})) \quad \text{where} \\ \alpha_u &= Ag((e_0, \alpha_0, \ldots, e_u)) \end{aligned}$$

We denote the set of all terminated runs of agent Ag in environment Env by $\mathcal{R}^*(Ag, Env)$, and the set of "least" terminated runs of Ag in Env (i.e., runs r in $\mathcal{R}^*(Ag, Env)$ such that no prefix of r is in $\mathcal{R}^*(Ag, Env)$) by $\mathcal{R}(Ag, Env)$. Hereafter, when we refer to "terminated runs" it should be understood that we mean "least terminated runs".

Tasks

We build agents in order to carry out *tasks* for us. We need to be able to describe the tasks that we want agents to carry out on our behalf; in other words, we need to *specify* them. In this paper, we shall be concerned with tasks that are specified via a predicate Ψ over runs:

$$\Psi : \mathcal{R} \to \{t, f\}.$$

The idea is that $\Psi(r) = t$ means that the run r satisfies specification Ψ, whereas $\Psi(r) = f$ means that run r does not satisfy the specification.

Given a complexity class $\mathcal{C}$, we say that specification Ψ is in $\mathcal{C}$ if the runs that satisfy Ψ form a language that can be recognised in $\mathcal{C}$. In this paper, we will focus primarily on two types of specification: those that are in the class Σ_u^p (for some $u \in \mathbb{N}$), and those that are PSPACE-complete. (To be accurate, in the first case we consider task specifications that are *strictly* in Σ_u^p, that is, in $\Sigma_u^p \setminus \Sigma_{u-1}^p$. For convenience, we define $\Sigma_u^p = \emptyset$ for $u < 0$; thus a task specification is in Σ_0^p if it is in $\Sigma_0^p \setminus \emptyset = \text{P} \setminus \emptyset = \text{P}$.) In both cases, we assume that a specification Ψ is represented as a Turing machine T_Ψ that accepts just those runs which satisfy the specification. Any Σ_u^p specification can be represented as an alternating Turing machine that operates in polynomial time using at most u alternations, and similarly, any PSPACE-complete specification can be represented as an polynomial time alternating Turing machine (which must therefore use at most a polynomial number of alternations).

3 Agent Verification

We can now state the agent verification problem:

> AGENT VERIFICATION
> *Given*: Environment Env, agent Ag, and task specification Ψ.
> *Answer*: "Yes" if every terminated run of Ag in Env satisfies specification Ψ, "No" otherwise.

This decision problem amounts to determining whether the following second-order formula is true:

$$\forall r \in \mathcal{R}(Ag, Env) \text{ we have } \Psi(r) = t.$$

The complexity of this problem will clearly be determined in part by the complexity of the task specification Ψ. If Ψ is undecidable, for example, then so is the corresponding verification problem. However, for more reasonable task specifications, we can classify the complexity of the verification problem. We focus our attention on the polynomial hierarchy of classes Σ_u^p (where $u \in \mathbb{N}$) [7, pp424–425] and the class PSPACE of task specifications that can be decided in polynomial space [7, pp455-489]. (Recall that in the first case, we are concerned with task specifications that are *strictly* in Σ_u^p — see the discussion above.)

Theorem 1. *The agent verification problem for Σ_u^p task specifications is Π_{u+1}^p-complete.*

Proof. To show that the problem is in Π_{u+1}^p, we sketch the design of a Turing machine that decides the problem by making use of a single universal alternation, and invoking a Σ_u^p oracle. The machine takes as input an agent Ag, an environment Env, and a Σ_u^p task specification Ψ:

1. *universally select all runs $r \in \mathcal{R}(Ag, Env)$;*
2. *for each run r, invoke the Σ_u^p oracle to determine the value of $\Psi(r)$;*
3. *accept if $\Psi(r) = t$, reject otherwise.*

Clearly this machine decides the problem in

$$\Pi_1^{p\,\Sigma_u^p} = co\text{-}\mathrm{NP}^{\Sigma_u^p} = co\text{-}\Sigma_{u+1}^p = \Pi_{u+1}^p.$$

To show the problem is Π_{u+1}^p-complete, we reduce the problem of determining whether QBF$_{u+1,\forall}$ formulae are true to agent verification [7, p428]. An instance of QBF$_{u+1,\forall}$ is given by a quantified Boolean formula with the following structure:

$$\forall \overline{x_1}\, \exists \overline{x_2}\, \forall \overline{x_3}\, \cdots\, Q_{u+1} \overline{x_{u+1}}\;\; \chi(\overline{x_1}, \ldots, \overline{x_{u+1}}) \tag{1}$$

in which:

- *the quantifier Q_{u+1} is "$\forall$" if u is even and "$\exists$" otherwise;*
- *each $\overline{x_i}$ is a finite collection of boolean variables; and*
- *$\chi(\overline{x_1}, \ldots, \overline{x_{u+1}})$ is a propositional logic formula over the Boolean variables $\overline{x_1}, \ldots, \overline{x_{u+1}}$.*

Such a formula is true if for all assignments that we can give to Boolean variables x_1, we can assign values to Boolean variables x_2, ..., such that $\chi(\overline{x_1}, \ldots, \overline{x_{u+1}})$ is true. Here is an example of a QBF$_{2,\forall}$ formula:

$$\forall x_1 \exists x_2 [(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)] \tag{2}$$

Formula (2) in fact evaluates to false. (If x_1 is false, there is no value we can give to x_2 that will make the body of the formula true.)

Given a QBF$_{u+1,\forall}$ formula (1), we produce an instance of agent verification as follows. First, let $\overline{x_1} = x_1^1, x_1^2, \ldots, x_1^m$ be the outermost collection of universally quantified variables. For each of these variables x_1^i, we create two possible environment states, $e_{x_1^i}$ and $e_{\neg x_1^i}$, corresponding to a valuation of true or false respectively to the variable x_1^i. We create an additional environment state e_0, *to serve as the initial state. The environment only allows the agent to perform the action α_0, and the environment responds to the ith performance of action α_0 with either $e_{x_1^i}$ or $e_{\neg x_1^i}$. After m performances of α_0 the run terminates. In this way, each run determines a valuation for the universally quantified variables $\overline{x_1} = x_1^1, x_1^2, \ldots, x_1^m$; the set of all runs corresponds to the set of all possible valuations for these variables. Given a run r, we denote by $\chi(\overline{x_1}, \ldots, \overline{x_{u+1}})[r/\overline{x_1}]$ the Boolean formula obtained from $\chi(\overline{x_1}, \ldots, \overline{x_{u+1}})$ by substituting for each variable x_1^i the valuation (either true or false) made in run r to x_1^i.*

Finally, we define a task specification Ψ as follows:

$$\Psi(r) = \begin{cases} t & \text{if the } \Sigma_u^p \text{ formula } \exists\overline{x_2}\,\forall\overline{x_3}\cdots Q_{u+1}\overline{x_{u+1}}\;\chi(x_1, \ldots, x_{u+1})[r/\overline{x_1}] \text{ is true} \\ f & \text{otherwise.} \end{cases}$$

Clearly, the input formula (1) will be true if the agent succeeds in satisfying the specification on all runs. Since the reduction is polynomial time, we are done.

We can also consider more expressive task specification predicates: those that are PSPACE-complete. In fact, it turns out that having a PSPACE-complete task specification predicate does not add to the complexity of the overall problem.

Theorem 2. *The agent verification problem for* PSPACE*-complete task specifications is also* PSPACE*-complete.*

Proof. PSPACE*-hardness is obvious from the complexity of the task specification. To show membership of* PSPACE *we first show that the complement of the agent verification problem for* PSPACE*-complete task specifications is in* PSPACE*. The complement of the agent verification problem involves showing that some run does not satisfy the specification, i.e., that there is a run $r \in \mathcal{R}(Ag, Env)$ of agent Ag in environment Env such that $\Psi(r) = f$. The following non-deterministic algorithm decides the problem:*

1. *guess a run $r \in \mathcal{R}(Ag, Env)$, of length at most $|E \times Ac|$;*
2. *verify that $\Psi(r) = f$.*

This algorithm runs in

$$\text{NPSPACE}^{\Pi_1^p} = \text{PSPACE}^{\Pi_1^p} = \text{PSPACE}.$$

*Hence the verification problem is in co-*PSPACE*. But co-*PSPACE=PSPACE *[7, p142], and so we conclude that the agent verification problem for* PSPACE*-complete task specifications is in* PSPACE*.*

An obvious question is whether "simpler" environments can reduce the complexity of the verification problem. An obvious candidate is *deterministic* environments. An environment is deterministic if every element in the range of τ is either a singleton or the empty set. For such environments, the result of performing any action is at most one environment state.

Theorem 3. *Let $\mathcal{C}$ be a complexity class that is closed under polynomial time reductions. Then the verification problem for deterministic environments and task specifications that are in $\mathcal{C}$ is also in $\mathcal{C}$.*

Proof. *Generate the run r of the agent in the environment (which can be done with $O(|E \times Ac|)$ calls on the Turing machine representing the environment, each of which requires at most time polynomial in $|E \times Ac|$), and then verify that $\Psi(r) = t$ (which can be done in $\mathcal{C}$).*

Another obvious simplification is to consider *Markovian* environments: an environment is Markovian, or history independent, if the possible next states of the environment only depend on the current state and the action performed. Although they are intuitively simpler than their non-Markovian counterparts, from the verification point of view, Markovian environments are in fact no better than non-Markovian environments.

Theorem 4. *The agent verification problem for Markovian environments and Σ_u^p strategy specifications is Π_{u+1}^p-complete.*

Proof. *The proof of Theorem 1 will suffice without alteration: the critical point to note is that the environment generated in the reduction from* $\text{QBF}_{u+1,\forall}$ *is Markovian.*

4 Types of Tasks

In this section, we use the results established in the preceding section to analyse the complexity of verification for three classes of task specifications: achievement and maintenance tasks, and tasks specified as arbitrary Boolean combinations of achievement and maintenance tasks.

4.1 Achievement and Maintenance Tasks

There are two obvious special types of tasks that we need to consider [10]: *achievement tasks* and *maintenance tasks*. Intuitively, an achievement task is specified by a number of “goal states”; the agent is required to bring about one of these goal states. Note that we do not care *which* goal state is achieved — all are considered equally good. Achievement tasks are probably the most commonly studied form of task in artificial intelligence. An achievement task is specified by some subset $\mathcal{G}$ (for “good” or “goal”) of environment states E. An agent is *successful* on a particular run if it is guaranteed to bring about one of the states $\mathcal{G}$, that is, if every run of the agent in the environment results in one of the states $\mathcal{G}$. We say an agent *Ag* succeeds in an environment *Env* if every run of the agent in that environment is successful. An agent thus succeeds in an environment if it can guarantee to bring about one of the goal states.

Example 1. Consider the environment whose state transformer function is illustrated by the graph in Figure 1. In this environment, an agent has just four

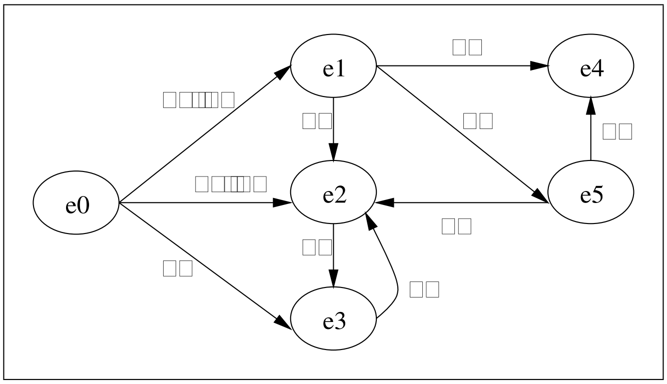


Fig. 1. The state transitions of an example environment: Arcs between environment states are labelled with the sets of actions corresponding to transitions. Note that this environment is *history dependent*, because agents are not allowed to perform the same action twice. So, for example, if the agent reached state e_2 by performing α_0 then α_2, it would not be able to perform α_2 again in order to reach e_3.

available actions (α_1 to α_4 respectively), and the environment can be in any of six states (e_0 to e_5). History dependence in this environment arises because the agent is not allowed to execute the same action twice. Arcs between states in Figure 1 are labelled with the actions that cause the state transitions — note that the environment is non-deterministic. Now consider the achievement tasks determined by the following goal sets:

- $\mathcal{G}_1 = \{e_2\}$
  An agent can reliably achieve $\mathcal{G}_1$ by performing α_1, the result of which will be either e_1, e_2, or e_3. If e_1 results, the agent can perform α_0 to take it to e_5 and then α_2 to take it to e_2. If e_3 results, it can simply perform α_0.
- $\mathcal{G}_2 = \{e_3\}$
  There is no agent that can be guaranteed to achieve $\mathcal{G}_2$. If the agent performs α_1, then any of e_1 to e_3 might result. In particular, if e_1 results, the agent can only get to e_3 by performing α_2 twice, which is not allowed.

Just as many tasks can be characterised as problems where an agent is required to bring about some state of affairs, so many others can be classified as problems where the agent is required to *avoid* some state of affairs, that is, to maintain some invariant condition. As an extreme example, consider a nuclear reactor agent, the purpose of which is to ensure that the reactor never enters a “meltdown” state. Somewhat more mundanely, we can imagine a software agent, one of the tasks of which is to ensure that a particular file is never simultaneously open for both reading and writing. We refer to such tasks as *maintenance* tasks.

A maintenance task is formally defined by a set $\mathcal{B} \subseteq E$ that we refer to as the “bad”, or “failure” states — these are the environment states that the agent must avoid. An agent Ag in environment Env is deemed successful with respect to such a maintenance specification if no state in $\mathcal{B}$ occurs on any run in $\mathcal{R}(Ag, Env)$.

Example 2. Consider again the environment in Figure 1, and the maintenance tasks defined by the following bad sets:

– $\mathcal{B}_1 = \{e_5\}$
  There is clearly an agent that can avoid e_5. After the agent performs its first action (either α_0 or α_1), one of the three states e_1 to e_3 will result. If the state that results is e_1, then the agent can perform α_2, after which either e_4 or e_2 will result; there will be no allowable moves in either case. If the state that results is e_2, then the agent can only perform α_2, which will transform the environment to e_4. The only allowable move will then be α_0 (if this has not already been performed — if it has, then there are no allowable moves); if the agent performs α_0, then environment state e_2 will result, from where there will be no allowable moves. Finally, if the state that results is e_3, then the agent can only perform α_0 and then α_2, which returns the environment to state e_3 from where there are no allowable moves.
– $\mathcal{B}_1 = \{e_2\}$
  No agent can be guaranteed to avoid e_2. Whether or not the first action is α_0 or α_1, it is possible that e_2 will result.

Given a an achievement task specification with good set $\mathcal{G}$, or a maintenance task specification with bad set $\mathcal{B}$, it should be clear that the corresponding specifications $\Psi_{\mathcal{G}}$ and $\Psi_{\mathcal{B}}$ will be decidable in (deterministic) polynomial time, i.e., in Σ_0^p. For example, to determine whether a run $r \in \mathcal{R}$ satisfies $\Psi_{\mathcal{G}}$, we simply check whether any member of $\mathcal{G}$ occurs on r. We can therefore apply Theorem 1 to immediately conclude the following.

Corollary 1. *The agent verification problem for achievement and maintenance tasks is Π_1^p-complete (i.e., co-*NP*-complete).*

4.2 Boolean Task Specifications

We can also consider a very natural extension to achievement and maintenance tasks, in which tasks are specified as *arbitrary Boolean combinations* of achievement and maintenance tasks. We specify such tasks via a formula of propositional logic χ. Each primitive proposition p that occurs in χ corresponds to an achievement goal, and hence a set $E_p \subseteq E$ of environment states. A negated proposition $\neg p$ corresponds to a maintenance task with bad states $E_{\neg p} \subseteq E$. From such primitive propositions, we can build up more complex task specifications using the Boolean connectives of propositional logic. To better understand the idea, consider the following example.

Example 3. Recall the environment in Figure 1, and suppose that we allow three primitive proposition letters to be used: p_1, p_2, p_3, where p_1 corresponds to environment states $\{e_2\}$, p_2 corresponds to $\{e_3\}$, and p_3 corresponds to $\{e_1\}$. Consider the following task specifications:

- p_1
  This is an achievement task with goal states $\{e_2\}$. Since p_1 corresponds to $\{e_2\}$, an agent can reliably achieve p_1 by performing α_1, the result of which will be either e_1, e_2, or e_3. If e_1 results, the agent can perform α_0 to take it to e_5 and then α_2 to take it to e_2. If e_3 results, it can simply perform α_0.
- $\neg p_1$
  This is essentially a maintenance task with bad states $\{e_2\}$. Since the agent must perform either α_0 or α_1, no agent can guarantee to avoid e_2.
- $p_1 \vee p_2$
  Since there is an agent that can be guaranteed to succeed with task p_1, there is also an agent that can be guaranteed to succeed with task $p_1 \vee p_2$.
- $p_1 \wedge p_2$
  This task involves achieving *both* e_2 and e_3. There is no agent that can be guaranteed to succeed with this task.
- $p_1 \wedge (p_2 \vee p_3)$
  This task involves achieving e_2 and either e_3 or e_1. There exists an agent that can successfully achieve this task.
- $(p_1 \wedge \neg p_2) \vee (p_2 \wedge \neg p_3)$
  This task involves either achieving e_2 and not e_3 or else achieving e_3 and not e_1. Since there exists an agent that can achieve e_2 and not e_3, there exists an agent that can succeed with this task.

Let us now formalise these ideas. We start from a set $\Phi = \{p_1, \ldots, p_n\}$ of n Boolean variables. A propositional formula $\chi(\Phi_n)$ over Φ_n is inductively defined by the following rules:

1. $\chi(\Phi_n)$ consists of single literal (x or $\neg x$ where $x \in \Phi_n$);
2. $\chi(\Phi_n) = \chi_1(\Phi_n)\ \theta\ \chi_2(\Phi_n)$, where $\theta \in \{\vee, \wedge\}$ and χ_1 and χ_2 are propositional formulae.

Let $f_{\chi(\Phi_n)}$ be the Boolean logic function (of n variables) represented by $\chi(\Phi_n)$. Let $S = \langle E_1, E_2, \ldots, E_n \rangle$ be an ordered collection of *pair-wise disjoint* subsets of E (note that S does *not* have to be a partition of E) and $\chi(\Phi_n)$ be a non-trivial formula. If $r \in \mathcal{R}$ is a run, then the instantiation, $\beta(r) = \langle b_1, b_2, \ldots, b_n \rangle$ of Boolean values to Φ_n *induced by* r is defined by:

$$b_i = \begin{cases} 1 & \text{if } r \text{ contains some state } e \in E_i \\ 0 & \text{otherwise.} \end{cases}$$

In other words, a proposition $p_i \in \Phi_n$ is defined to be true with respect to a run r if one of the states in E_i occurs in r; otherwise, p_i is defined to be false.

A run, r, *succeeds with respect to* the formula χ if $f_\chi(\beta(r)) = 1$. An agent Ag satisfies the Boolean task specification χ in environment Env if $\forall r \in \mathcal{R}(Ag, Env)$ we have $f_\chi(\beta(r)) = 1$. Given a run $r \in \mathcal{R}$ and a Boolean task specification $\chi(\Phi_n)$ the problem of determining whether $f_\chi(\beta(r)) = 1$ is decidable in deterministic polynomial time[1]. So, by Theorem 1, we can conclude:

[1] The proof is by an induction on the structure of $\chi(\Phi_n)$, with the inductive base where $\chi(\Phi_n)$ is a positive literal (an achievement task) or a negative literal (a maintenance task).

Corollary 2. *The agent verification problem for Boolean task specifications is Π_1^p-complete.*

5 Related Work

Verification is, of course, a major topic in theoretical computer science and software engineering (see, e.g., [2]). Many techniques have been developed over the past three decades with the goal of enabling efficient formal verification. These approaches can be broadly divided into two categories: *deductive* verification and *model checking*. Deductive approaches trace their origins to the work of Hoare and Floyd on axiomatizing computer programs [4]. Deductively verifying that a system S satisfies some property Ψ, where Ψ is expressed as a formula of some logic $\mathcal{L}$, involves first generating the $\mathcal{L}$-theory $Th(S)$ of S and then using a proof system $\vdash_{\mathcal{L}}$ for $\mathcal{L}$ to establish that $Th(S) \vdash_{\mathcal{L}} \Psi$. The theoretical complexity of the proof problem for any reasonably powerful language $\mathcal{L}$ has been a major barrier to the automation of deductive verification. For example, a language widely proposed for the deductive verification of reactive systems is first-order temporal logic [5,6]; but proof in first-order temporal logic is not even semi-decidable.

Perhaps closer to our view is the more recent body of work on verification of finite state systems by model checking [3]. The idea is that a system S to be verified can be represented as a model M_S for a branching temporal logic. If we express the specification Ψ as a formula of the temporal logic, then model checking amount to showing that Ψ is valid in M_S: this can be done in deterministic polynomial time for the branching temporal logic CTL [3, p38], wherein lies much of the interest in model checking as a practical approach to verification.

The main difference between our work and that on model checking is that we explicitly allow for a system to be composed of an environment part and an agent part, and our environments are assumed to be much less compliant than is usually the case in model checking (where environments are implicitly both deterministic and Markovian, which allows them to be represented as essentially directed graphs). Some researchers have begun to examine the relationship between model checking and agents in more detail [8,1], and it would be interesting to examine the relationship of our work to this formally.

6 Conclusions

In this paper, we investigated the complexity of the agent verification problem, that is, the problem of showing that a particular agent, when placed in a particular environment, will satisfy some particular task specification. We established two main complexity results for this problem, which allow us to classify the complexity of the problem in terms of the complexity of deciding whether the task specification has been satisfied on any given run of the agent. We first proved that for tasks with specifications that are in Σ_u^p, the corresponding agent verification problem is Π_{u+1}^p-complete; we then showed that for PSPACE-complete task specifications, the corresponding verification problem is also PSPACE-complete.

We then used these results to analyse the computational complexity of various common kinds of tasks, including achievement and maintenance tasks and tasks specified as arbitrary Boolean combinations of achievement and maintenance tasks.

There are several important avenues for future work. The first is on the relationship of the verification problem to model checking, as described above. The second is on *probabilistic* verification: Given agent Ag, environment Env, specification Ψ, and probability p, does Ag in Env satisfy Ψ with probability at least p?

Acknowledgments

This research was supported by the EPSRC under grant GR/R60836 ("Algorithmics for agent design and verification").

References

1. M. Benerecetti, F. Giunchiglia, and L. Serafini. Model checking multi-agent systems. *Journal of Logic and Computation*, 8(3):401–424, 1998.
2. R. S. Boyer and J. S. Moore, editors. *The Correctness Problem in Computer Science*. The Academic Press: London, England, 1981.
3. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press: Cambridge, MA, 2000.
4. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, 1969.
5. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag: Berlin, Germany, 1992.
6. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems — Safety*. Springer-Verlag: Berlin, Germany, 1995.
7. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley: Reading, MA, 1994.
8. A. S. Rao and M. P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 318–324, Chambéry, France, 1993.
9. S. Russell and D. Subramanian. Provably bounded-optimal agents. *Journal of AI Research*, 2:575–609, 1995.
10. M. Wooldridge. The computational complexity of agent design problems. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 341–348, Boston, MA, 2000.
11. M. Wooldridge and P. E. Dunne. Optimistic and disjunctive agent design problems. In Y. Lespérance and C. Castelfranchi, editors, *Proceedings of the Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, 2000.

A Goal-Based Organizational Perspective on Multi-agent Architectures

Manuel Kolp[1], Paolo Giorgini[2], and John Mylopoulos[3]

[1]IAG - Information Systems Research Unit - University of Louvain,
1 Place des Doyens, B-1348 Louvain-La-Neuve, Belgium, tel.: 32-10 47 83 95,
kolp@isys.ucl.ac.be
[2]Department of Mathematics - University of Trento,
4 via Sommarive, I-38100, Trento, Italy, tel.: 39-0461-88 2052,
pgiorgini@science.unitn.it
[3]Department of Computer Science - University of Toronto,
6 King's College Road M5S 3H5, Toronto, Canada, tel.: 1-416-978 5180,
jm@cs.toronto.edu

Abstract. A Multi-Agent System (MAS) is an organization of coordinated autonomous agents that interact in order to achieve common goals. Considering real world organizations as an analogy, this paper proposes architectural styles for MAS which adopt concepts from organization theory and strategic alliances literature. The styles are intended to represent a macro-level architecture of a MAS, and they are modeled using the *i** framework which offers the notions of actor, goal and actor dependency for modeling multi-agent settings. The styles are also specified as metaconcepts in the Telos modeling language. Moreover, each style is evaluated with respect to a set of software quality attributes, such as predictability and adaptability. The paper also explores the adoption of micro-level patterns proposed elsewhere in order to give a finer-grain description of a MAS architecture. These patterns define how goals assigned to actors participating in an organizational architecture will be fulfilled by agents. An e-business example illustrates both the styles and patterns proposed in this work. The research is being conducted within the context of *Tropos*, a comprehensive software development methodology for agent-oriented software.

1 Introduction

Multi-Agent System (MAS) architectures can be considered as organizations (see e.g., [6, 7, 15]) composed of *autonomous* and *proactive* agents that interact and cooperate with one another in order to achieve common or private goals. In this paper, we propose to use real world organizations as a metaphor in order to offer a set of generic architectural patterns (or, *styles*) for distributed and open systems, such as MAS. These styles have been adopted from the organization theory and strategic alliances literature [12]. The styles are modeled using the strategic dependency model of *i** [22], and they are further specified in the Telos modeling language [14]. We also present multi-agent patterns to design MAS architectures at a finer-grain.

To illustrate the use of these styles and patterns, we use as example a (fictitious) *Media Shop*. This is a store selling and shipping different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes, and the like. *Media Shop* customers (on-site or remote) can use a periodically updated catalogue describing available media items to specify their order. *Media Shop* is supplied latest releases and in-catalogue items by *Media Supplier*. To increase market share, *Media Shop* has decided to open up a B2C retail sales front on the internet. With the new setup, a customer can order *Media Shop* items in person, by phone, or through the internet. The system has been named *Medi@* and is available on the world-wide-web using communication facilities provided by *Telecom Co*. It also uses financial services supplied by *Bank Co.*, which specializes on on-line transactions.

This research is being conducted within the context of the Tropos project [2]. Tropos adopts ideas from MAS technologies, mostly to define the detailed design and implementation phase, and ideas from requirements engineering, where agents/actors and goals have been used heavily for early requirements analysis [4, 22]. In particular, Tropos is founded on Eric Yu's *i** modeling framework which offers actors (agents, roles, or positions), goals, and actor dependencies as primitive concepts for modelling an application during early requirements analysis. The key premise of the project is that actors and goals can be used as fundamental concepts for analysis and design during *all phases of software development*, not just requirements analysis.

Section 2 introduces the macro level catalogue of organization-inspired architectural styles, and proposes a set of software quality attributes for evaluating architectural alternatives. Section 3 introduces the micro level catalogue of goal-based multi-agent patterns for finer-grain design of an organizational architecture. Section 4 presents fragments of an e-business case study to illustrate the use of styles and patterns proposed in the paper. Finally, Section 5 summarizes the contributions of the paper and points to further work.

2 Organizational Styles

Organizational theory [13, 19] and strategic alliances [9, 20, 21] study alternative styles for (business) organizations. These styles are used to model the coordination of business stakeholders -- individuals, physical or social systems -- to achieve common goals. We propose a macro level catalogue adopting (some of) these styles for designing MAS architectures.

A strategic dependency model is a graph, where each node represents an actor (an agent, position, or role) and each link between two actors indicates that one actor depends on another for a goal to be fulfilled, a task to be carried out, or a resource to be made available. We call the depending actor of a dependency the *depender* and the actor who is depended upon the *dependee*. The object around which the dependency centers (goal, task or resource) is the *dependum*. The model distinguishes among four types of dependencies -- goal-, task-, resource-, and softgoal-dependency -- based on the type of freedom that is allowed in the relationship between depender and dependee. Softgoals are distinguished from goals because they do not have a formal definition, and are amenable to a different (more qualitative) kind of analysis [3].

For instance, in Figure 1, the Middle Agency and Support actors depend on the Apex for strategic management. Since the goal Strategic Management does not have a precise description, it is represented as a softgoal (cloudy shape). The Middle Agency depends on Coordination and Support respectively through goal dependencies Control and Logistics represented as oval-shaped icons. Likewise, the Operational Core actor is related to the Coordination and Support actors through the Standardize task dependency and the Non-operational Service resource dependency, respectively.

The **structure-in-5** (s-i-5) style (Figure 1) consists of five typical strategic and logistic components found in many organizations. At the base level one finds the Operational Core where the basic tasks and operations -- the input, processing, output and direct support procedures associated with running the system -- are carried out. At the top lies the Apex composed of strategic executive actors. Below it, sit the control/standardization, management components and logistics: Coordination, Middle Agency and Support, respectively. The Coordination component carries out the tasks of standardizing the behavior of other components, in addition to applying analytical procedures to help the system adapt to its environment. Actors joining the Apex to the Operational Core make up the Middle Agency.

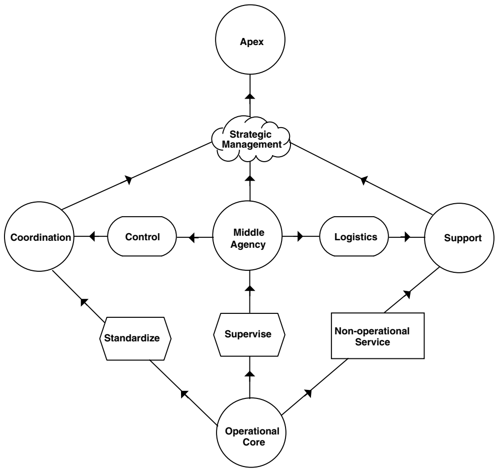


Fig. 1. Structure-in-5

The Support component assists the Operational Core for non-operational services that are outside the basic flow of operational tasks and procedures.

Figure 2 specifies the structure-in-5 style in Telos. Telos is a language intended for modeling requirements, design and implementation for software systems. It provides features to describe metaconcepts used to represent the knowledge relevant to a variety of worlds – subject, usage, system, development worlds - related to a software system. Our organizational styles are formulated as Telos metaconcepts, using heavily aggregation semantics, as proposed in [16].

`MetaClass` is a metametaclass with all metaclasses as instances. `ApexClass`, `CoordinationClass`, `MiddleAgencyClass`, `SupportClass`, and `OperationalCoreClass` are also metaclasses whose instances are actor classes of five different kinds, as indicated by their names. The structure-in-5 style is then a metaclass defined as an aggregation of five (*part*) metaclasses. Each of these five components exclusively belongs (exclusivePart) to the aggregate and their existence depends (dependentPart) on the existence of the composite. A structure-in-5 architecture specific to an application domain can now be defined as a Telos class, which instantiates StructureIn5MetaClass. Similarly, each structure-in-5 component specific to a particular application domain will be defined as a class, instance of one of the five StructureIn5Metaclass components.

```
        TELL CLASS StructureIn5Class IN MetaClass WITH
         attribute name: String
            part, exclusivePart, dependentPart
                  apex: ApexActorClass
                  coordination: CoordinationActorClass
                  middleAgency: MiddleAgencyActorClass
                  support: SupportActorClass
                  operationalCore: OperationalCoreClass
      END StructureIn5MetaClass
```

Fig. 2. Structure-in-5 in Telos

Figure 3 formulates in Telos one of these five structure-in-5 components: the Coordination actor. Dependencies are described following Telos specifications for *i** models. The Coordination actor is a metaclass, CoordinationMetaclass. According to Figure 1, the Coordination actor is the dependee of a task dependency StandardizeTask and a goal dependency ControlGoal, and the depender of a softgoal dependency StrategicManagementSoftGoal.

```
TELL CLASS CoordinationMetaclass IN MetaClass WITH
  attribute name: String
   taskDepended
        s:StandardizeTask
        WITH depender
           opCore: OperationalCoreClass
        END
   goalDepended
        c:ControlGoal
        WITH depender
           midAgency: MiddleAgencyClass
        END
   softgoalDepender
        s:StrategicManagementSoftGoal
        WITH dependee
           apex: ApexClass
        END
END CoordinationMetaclass
```

Fig. 3. Structure-in-5 coordination actor.

The **pyramid** (pyr) style is the well-known hierarchical authority structure exercised within organizational boundaries. Actors at the lower levels depend on and report to actors of higher levels. The crucial mechanism here is direct or indirect supervision by the apex. Managers and supervisors are then only intermediate actors routing strategic decisions and authority from the apex to the operating level. They can coordinate behaviors or take decisions on their own, but only at a local level. This style can be applied when designing simple distributed systems. Moreover, this style encourages dynamicity since coordination and decision mechanisms are direct and immediately identifiable. For applications which require a high degree of evolvability and modifiability, this is a good style to use. However, this style is not suitable for complex distributed systems requiring many kinds of agents and supervision relationships. On the other hand, it can be used by such systems to manage and resolve crisis situations. For instance, a complex multi-agent system faced with a non-authorized intrusion from external agents could dynamically reconfigure itself into a pyramid structure in order to resolve the security problem in an effective way.

The **joint venture** (jo-ve) style (Figure 4a) involves agreement between two or more principal partners to obtain the benefits of larger scale operation, with only partial investment and lower maintenance costs. This is accomplished by delegating authority to a specific Joint Management actor who coordinates tasks and manages sharing of knowledge and resources. Each principal partner can manage and control its own operations on a local dimension, but also interact directly with other principal partners to provide and receive services, data and knowledge. However, the strategic operation and coordination of such a system and its partner actors on a global dimension are only ensured by the Joint Management actor. Outside the joint venture, secondary partners supply services or support tasks for the organization core.

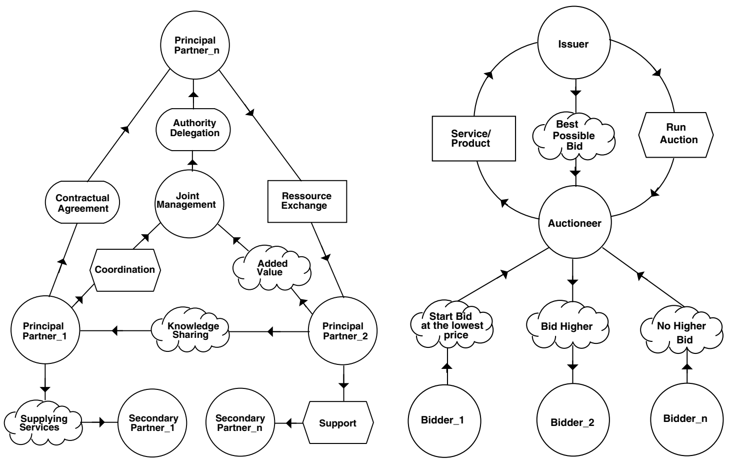


Fig. 4. Joint Venture (a) and bidding (b)

The **bidding** (bidd) style (Figure 4b) is founded on competition mechanisms and actors behave as if they were taking part in an auction. The Auctioneer actor runs the whole show. It advertises the auction issued by the auction Issuer, receives bids from bidder actors and ensure communication and feedback with the auction Issuer. The auction Issuer is responsible for issuing the bidding.

The **arm's-length** (ar-le) style implies agreement between independent and competitive actors who are willing to join a partnership. Partners keep their autonomy and independence but act and put their resources and knowledge together to accomplish precise common goals. Authority is not delegated by any partner.

The **hierarchical contracting** (hi-co) style identifies coordinating mechanisms that combine arm's-length agreement features with aspects of pyramidal authority. Coordination here uses mechanisms with arm's-length (i.e., high independence) characteristics involving a variety of negotiators, mediators and observers. These work at different levels and handle conditional clauses, monitor and manage possible contingencies, negotiate and resolve conflicts and finally deliberate and take decisions. Hierarchical relationships, from the executive apex to the arm's-length contractors (top to bottom) restrict autonomy and underlie a cooperative venture between the contracting parties. Such, admittedly complex, contracting arrangements can be used to manage conditions of complexity and uncertainty deployed in high-cost-high-gain (high-risk) applications.

Table 1. : Correlation catalogue.

	s-i-5	pyr	jo-ve	bidd	ar-le	hi-co	co-op
Predictability	+	++	+	--	-		-
Security	+	++	+	--	--		-
Adaptability		+	++	++	+	+	++
Cooperativity	+	++	+	-	-	+	++
Competitivity	-	-	-	++	++	+	+
Availability	+	+	++	-	--	+	--
Failability-Tolerance		--		--	++		-
Modularity	++	-	+	++	+	+	--
Aggregability	++		++			+	

The **co-optation** (co-op) style involves the incorporation of representatives of external systems into the decision-making or advisory structure and behavior of a newly-created organization. By co-opting representatives of external systems, an organization is, in effect, trading confidentiality and authority for resource, knowledge assets and support.

The evaluation of these styles can be done with respect to desirable software quality attributes identified as relevant for distributed and open architectures such as multi-agent ones. For lack of space, we do not detail them here and refer to [12] where a full description of such attributes is presented. Table 1 summarizes correlations for our styles and the quality attributes: +, ++, -, -- respectively model partial/positive, sufficient/positive, partial/negative and sufficient/negative contributions [3].

3 Multi-agent Patterns

A further step in the architectural design of MAS consists of specifying how the goals delegated to each actor are to be fulfilled. For this step, designers can be guided by a catalogue of multi-agent patterns which offer a set of standard solutions. Design patterns have received considerable attention in Software Engineering [8, 17] and some of these could be adopted for MAS architectures. Unfortunately, they focus on object-oriented rather than agent-oriented systems.

In the area of MAS, some work has been done in designing agent patterns, e.g.,[1, 5, 11]. However, these contributions focus on agent communication, while we are interested in specifying how a goal is to be achieved at an organization level.

In the following we present a micro level catalogue of often-encountered multi-agent patterns in the MAS literature. In particular, some of the federated patterns introduced in [10, 23] will be used in Section 4.

A **broker** is an arbiter and intermediary accessing services from a provider to satisfy the request of a consumer. It is used in vertical integration and joint venture architectures.

A **matchmaker** (Figure 5a) locates a provider for a given consumer service request, and then lets the consumer interact directly with the provider. In this respect, matchmakers are different from brokers who directly handle all interactions between the consumer and the provider. This pattern is also useful for horizontal integration and joint venture architectures.

A **monitor** (Figure 5b) alerts a subscriber when certain events occur. This type of agent accepts subscriptions, requests notifications from subjects of interest, receives such notifications of events and alerts subscribers accordingly. The subject provides notifications of state changes as requested. The subscriber registers for notification of state changes to subjects, and receives notifications of changes. This pattern is used in horizontal contracting, vertical integration, arm's-length and bidding styles all of which require monitoring activities.

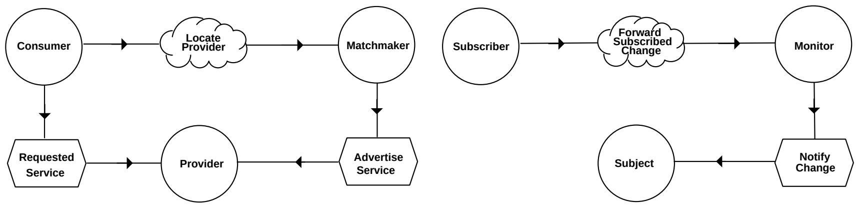


Fig. 5. Matchmacker (a) and monitor (b)

A **mediator** (Figure 6a) mediates interactions among different agents. An initiator addresses the mediator instead of asking directly another agent, the performer. A mediator has acquaintance models of other actors and coordinates cooperation among them. Conversely, other agents have an acquaintance model for the mediator. While a matchmaker simply matches providers with consumers, a mediator encapsulates interactions and maintains models of initiators and performers behaviors over time. It is used in pyramid, vertical integration and horizontal contracting styles because it underlies direct cooperation and encapsulation features reinforcing authority.

An **embassy** (Figure 6b) routes a service requested by an external agent to a local agent and hands back to the foreigner the response. If the request is granted, the external agent can submit messages to the embassy for translation. The content of each such message is translated in accordance with a standard ontology. Translated messages are forwarded to requested local agents. The results of the query are passed back out to the foreign agent, after translation. This pattern is useful for the structure-in-5, arm's-length, bidding and co-optation styles because it copes well with security issues that arise because of the competition mechanisms inherent to these styles.

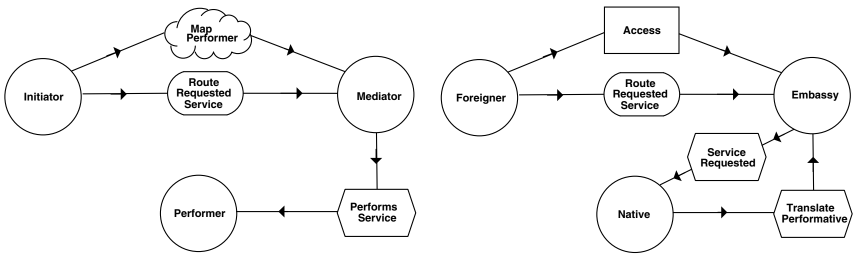


Fig. 6. Mediator (a) and embassy (b)

A **wrapper** incorporates a legacy system into a multi-agent system**.** The wrapper interfaces system agents with the legacy system by acting as a translator between them. This ensures that communication protocols are respected and the legacy system remains decoupled from the rest of the system. This pattern can be used in the co-optation style when one of the co-optated actor is a representing a legacy system.

The **Contract-Net** pattern selects an agent to which it assigns a task. This pattern includes a manager and any number of participants. The manager issues a request for proposals for a service to all participants and then accepts "proposals" that offer the service for a particular "cost". The manager selects one participant who performs the contracted work and informs the manager upon completion. This pattern is useful in the arm's-length, bidding and co-optation styles due to their competition mechanisms.

Table 2. : Agents' capabilities for the matchmaker pattern.

MATCHMAKER	
Agent	**Capabilities**
Customer	– Build a request to query the matchmaker – Handle with a services ontology – Query the matchmaker for a service – Find alternative matchmakers – Request a service to a provider – Manage possible provider failures – Monitor the provider s ongoing processes – Ask the provider to stop the requested service
Provider	– Handle with a services ontology – Advertise a service to the matchmaker – Withdraw the advertisement – Use an agenda for managing the requests – Inform the customer of the acceptance of the request service – Inform the customer of a service failure – Inform the customer r of success of a service
Matchmaker	– Update the local database – Handle with a services ontology – Use an agenda for managing the customer requests – Search the name of an agent for a service – Inform the customer of the unavailability of agents for a service

A detailed analysis of each pattern allows us to define a set of capabilities for agents playing a role in the pattern. Such capabilities are not exhaustive and concern exclusively agents activities related to each pattern's goal. For lack of space, we only present a set of capabilities for the matchmaker pattern (Table 2).

A capability states that an agent is able to act in order to achieve a given goal. In particular, for each capability the agent has (knows) a set of plans that may apply in different situations. A plan describes the sequence of actions to perform and the conditions under which the plan is applicable. At this stage we do not need to define the plans in detail. Instead, we simply specify that the agent needs to be capable of achieving in one or more ways a given goal. In the Tropos methodology plans are defined in the detail design phase. Sometimes several agents participating in a pattern need to have common capabilities. For instance, the capability handle with services ontology is common to all three agents of the Matchmaker pattern. This suggests a need for a capability pattern repository to be used during the implemention phase.

4 An e-Business Example

E-business systems are essential components of "virtual enterprises". By now, software architects have developed catalogues of web architectural styles. Common styles include the *Thin Web Client*, *Thick Web Client* and *Web Delivery.* These architectural styles focus on web concepts, protocols and underlying technologies but not on business processes nor on non-functional requirements for a given application. As a result, the organizational architecture styles are not described well within such a framework. Three software quality attributes are often important for an e-commerce architecture, according to [12]: *Security*, *Availability* and *Adaptability*.

To cope with these software quality attributes and select a suitable architecture for a system under design, we go through a means-ends analysis using the non functional requirements (NFRs) framework. This framework analyses desirable qualities by going through an iterative goal refinement and decomposition as shown in Figure 7. The analysis is intended to make explicit the space of alternatives for fulfilling the top-level qualities. The styles are represented as design decisions (saying, roughly, "make the architecture of the system respectively *pyramid, co-optation, joint venture, arm's-length*-based, …").

The evaluation results in contribution relationships from architectural styles to quality attributes, labeled "+", "++", "-", "--". Design rationale is represented by claims drawn as dashed clouds. These can represent priorities and other meta-information about the decision making process. Exclamation marks (! and !!) are used to mark priorities while a check-mark indicates an achieved quality, while and a cross indicates an un-achievable one.

In Figure 7, *Adaptability* has been AND-decomposed into *Dynamicity* and *Updatability*. For our e-commerce example, *dynamicity* is concerned with the use of generic mechanisms that allow web pages and user interfaces to be dynamically and easily changed. Indeed, information content and layout need to be frequently refreshed to give correct information to customers or simply follow fashion trends for marketing reasons. Using frameworks such as Active Server Pages (ASP) and Server Side Includes (SSI) to create dynamic pages goes some distance towards addressing this quality. *Updatability* is strategically important for the viability of an application. For our example, *Media Shop* employees have to update regularly the catalogue for inventory consistency. This type of analysis is to be carried out in turn for newly identified sub-qualities as well as for other top-level qualities such as *Security* and *Availability.*

Eventually, the analysis shown in Figure 6 allows us to choose the joint venture architectural style for our e-commerce example (qualities that have been achieved are marked with a check mark). The analysis uses the correlation catalogue shown in Table 1 and the top level qualities *Adaptability*, *Security* and *Availability*. These are respectively marked ++, +, ++ for the selected style. More fine-grain qualities have been identified during the decomposition process, such as *Integrity* (*Accuracy*, *Completeness*), *Usability*, *Response Time*, *Maintainability*, and more.

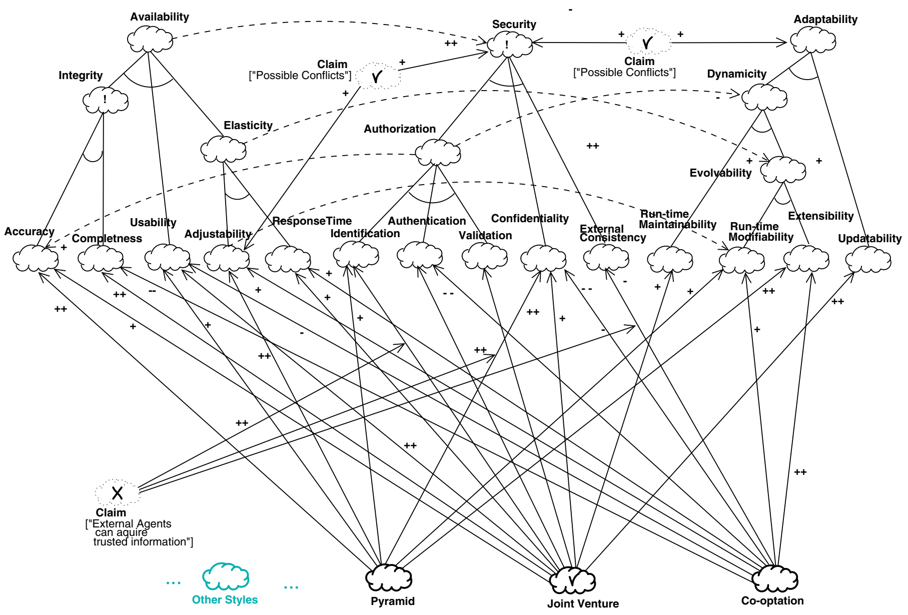


Fig. 7. Partial architecture evaluation for organizational styles.

Figure 8 offers a possible assignment of system responsibilities for our example, based on the joint venture style. The system consists of three principal partners, Store Front, Billing Processor and Back Store. Each of them delegates authority to, and is controlled and coordinated by, the joint management actor (Joint Manager) managing the system on a global dimension. Store Front interacts primarily with Customer and provides her with a usable front-end web application. Back Store keeps track of all web information about customers, products, sales, bills and other data of strategic importance to Media Shop. Billing Processor is in charge of the (secure) management of orders and bills, as well as other financial data. It is also in charge of interactions with Bank Cpy. Joint Manager manages all of the above, controlling *security*, *availability* and *adaptability* concerns.

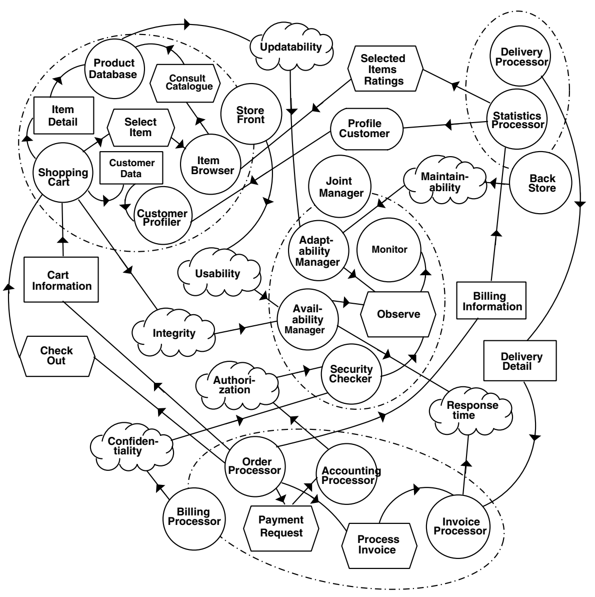


Fig. 8. An e-commerce system in joint venture architecture.

To accommodate the responsibilities of Store Front, we introduce Item Browser to manage catalogue navigation, Shopping Cart to select and custom items, Customer Profiler to track customer data and produce client profiles, and Product Database to manage media items information.

To cope with the identified software qualities (*security*, *availability* and *adaptability*), Joint Manager is further refined into four new sub-actors: Availability Manager, Security Checker and Adaptability Manager assume one of the main softgoals (and their subgoals). They are all monitored by Monitor. Further refinements are shown on Figure 8.

Figure 9 shows a possible use of some of the multi-agent patterns for designing in terms of agents the architecture of the e-business system shown in Figure 8. In particular, the broker pattern is applied to the Info Searcher, which satisfies requests of searching information by accessing Product Database. The Source Matchmaker applies

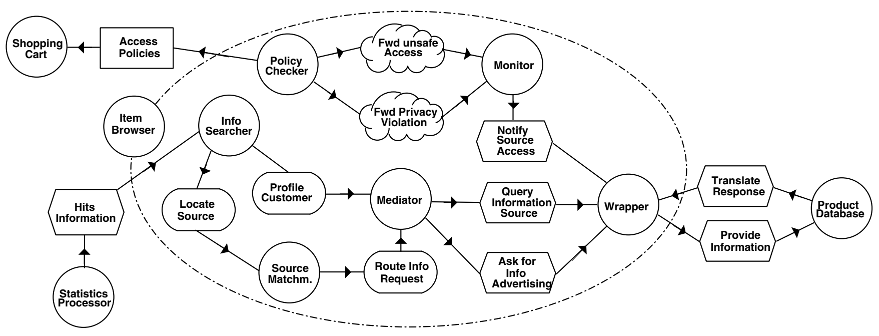


Fig. 9. Multi-agents patterns for item browser

the matchmaker pattern locating the appropriate source for the Info Searcher, and the monitor pattern is used to check both the correct use of the user data and the security for the sources accesses. Finally, the mediator pattern is applied to mediate the interaction among Info Searcher, Source Matchmaker and the Wrapper, while the wrapper pattern makes the interaction between Item Browser and Product Database possible. Other patterns can be applied as well. For instance, we could use the contract-net pattern to delegate to a wrapper the interaction with the Product Database, or the embassy to route the request of a wrapper to the Product Database.

5 Conclusions

Designers rely on styles, patterns, or idioms, to describe the architectures of their choice. We propose that MAS can be conceived as *organizations* of agents that interact to achieve common goals. This paper proposes a catalogue of architectural styles and agent patterns for designing MAS architectures at a macro- and micro-level. The proposed styles adopt concepts from organization theory and strategic alliances literature. The proposed patterns are based on earlier research within the agents community. The paper also includes an evaluation of software qualities that are relevant to these styles.

Future research directions include formalizing precisely the organizational styles and agent patterns that have been identified, as well as the sense in which a particular model is an instance of such a style and pattern. We also propose to compare and contrast them with classical software architectural styles and patterns proposed in the literature, and relate them to lower-level architectural components involving (software) components, ports, connectors, interfaces, libraries and configurations.

References

[1] Y. Aridor and D. B. Lange. "Agent Design Patterns: Elements of Agent Application Design" In *Proc. of the 2nd Int. Conf. on Autonomous Agents* (Agents'98), New York, USA, May 1998.

[2] J. Castro, M. Kolp and J. Mylopoulos. "A Requirements-Driven Development Methodology", In *Proc. of the 13th Int. Conf. on Advanced Information Systems Engineering* (CAiSE'01), Interlaken, Switzerland, June 2001.

[3] L. K. Chung, B. A. Nixon, E. Yu and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*, Kluwer Publishing, 2000.

[4] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal–directed Requirements Acquisition", *Science of Computer Programming, 20*, 1993, pp. 3-50.

[5] D.Deugo, F. Oppacher, J.Kuester, I. V. Otte. "Patterns as a Means for Intelligent Software Engineering". In *Proc. of the Int. Conf. of Artificial Intelligence* (IC-AI'99), Vol II, CSRA Press, 605-611, 1999.

[6] J. Ferber and O. Gutknecht."A meta-model for the analysis and design of organizations in multi-agent systems". In *Proc. of the 3rd Int. Conf. on Multi-Agent Systems* (ICMAS'98), June, 1998.

[7] M.S. Fox. "An organizational view of distributed systems". In *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):70-80, January 1981.

[8] E. Gamma., R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 1995
[9] B. Gomes-Casseres. *The alliance revolution : the new shape of business rivalry*, Harvard University Press, 1996.
[10] S. Hayden, C. Carrick, and Q. Yang. “Architectural Design Patterns for Multiagent Coordination”. In *Proc. of the 3rd Int. Conf. on Autonomous Agents* (Agents’99), Seattle, USA, May 1999.
[11] E. Kendall, P.V. Murali Krishna, C. V. Pathak, and C.B. Suersh. “Patterns of Intelligent and Mobile Agents”. In *Proc. of the* 2^{nd} *Int. Conf. on Autonomous Agents* (Agents’98), pages 92—98, New York, May 1998.
[12] M. Kolp, J. Castro and J. Mylopoulos, “A Social Organization Perspective on Software Architectures”. In *Proc. of the First Int. Workshop From Software Requirements to Architectures* (STRAW'01), Toronto, May 2001.
[13] H. Mintzberg, *Structure in fives : designing effective organizations*, Prentice-Hall, 1992.
[14] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis: “Telos: Representing Knowledge About Information Systems”. In *ACM Trans. Info. Sys.*, 8 (4), Oct. 1990, pp. 325 – 362.
[15] T.W. Malone. “Organizing Information Processing Systems: Parallels Between Human Organizations and Computer Systems”. In W. Zachry, S. Robertson and J. Black, eds. *Cognition, Cooperation and Computation*, Norwood, NJ: Ablex, 1988.
[16] R. Motschnig-Pitrik, “The Semantics of Parts Versus Aggregates in Data/Knowledge Modeling”, *Proc. of the 5th Int. Conf. on Advanced Information Systems Engineering* (CAiSE’93), Paris, June 1993, pp 352-372.
[17] W. Pree. *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, 1995.
[18] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.
[19] W. Richard Scott. *Organizations: rational, natural, and open systems*, Prentice Hall, 1998.
[20] L. Segil. *Intelligent business alliances : how to profit using today's most important strategic tool*, Times Business, 1996.
[21] M.Y. Yoshino and U. Srinivasa Rangan. *Strategic alliances : an entrepreneurial approach to globalization*, Harvard Business School Press, 1995.
[22] E. Yu. Modelling *Strategic Relationships for Process Reengineering*, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.

$\mathcal{MINERVA}$ - A Dynamic Logic Programming Agent Architecture

João Alexandre Leite, José Júlio Alferes, and Luís Moniz Pereira

Centro de Inteligência Artificial (CENTRIA)
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
`{jleite | jja | lmp}@di.fct.unl.pt`

Abstract. The agent paradigm, commonly implemented by means of imperative languages mainly for reasons of efficiency, has recently increased its influence in the research and development of computational logic based systems. Since efficiency is not always the crucial issue, but clear specification and correctness is, *Logic Programming* and *Non-monotonic Reasoning* have been brought back into the spotlight. To this accrues the recent significant improvements in the efficiency of *Logic Programming* implementations for *Non-monotonic Reasoning*.
This paper presents an overall description of $\mathcal{MINERVA}$, an agent architecture and system designed with the intention of providing a common agent framework based on the unique strengths of *Logic Programming*, to allow for the combination of several non-monotonic knowledge representation and reasoning mechanisms developed in recent years. In [10], the semantics of the multi-dimensional structure and combination of the evolving societal knowledge of agents in described and discussed in detail.

1 Introduction

The *Logic Programming (LP)* paradigm provides a well-defined, general, integrative, encompassing, and rigorous framework for systematically studying computation, be it syntax, semantics, procedures, or attending implementations, environments, tools, and standards. *LP* approaches problems, and provides solutions, at a sufficient level of abstraction so that they generalize from problem domain to problem domain. This is afforded by the nature of its very foundation in logic, both in substance and method, and constitutes one of its major assets. To this accrues the recent significant improvements in the efficiency of *Logic Programming* implementations for *Non-monotonic Reasoning* [11,14,16]. Besides allowing for a unified declarative and procedural semantics, eliminating the traditional wide gap between theory and practice, the use of several and quite powerful results in the field of non-monotonic extensions to *LP*, such as belief revision, inductive learning, argumentation, preferences, abduction, etc.[12,13] can represent an important composite added value to the design of rational agents. These results, together with the improvement in efficiency, allow the referred mustering of *Logic Programming* and *Non-monotonic Reasoning* to accomplish a feliticious combination between reactive and rational behaviours of agents, the *Holy Grail* of modern *Artificial Intelligence*, whilst preserving clear and precise specification enjoyed by declarative languages.

Until recently, *LP* could be seen as a good representation language for static knowledge. If we are to move to a more open and dynamic environment, typical of the agency paradigm, we need to consider ways of representing and integrating knowledge from different sources which may evolve in time. To solve this limitation, the authors, with others, first introduced *Dynamic Logic Programming (DLP)* [1]. There, they studied and defined the declarative and operational semantics of sequences of logic programs (or dynamic logic programs). Each program in the sequence contains knowledge about some given state, where different states may, for example, represent different time periods or different sets of priorities. In [9] the *DLP* paradigm was then generalized in order to allow, not only for sequences of logic programs, but for collections of programs structured by acyclic digraphs (DAGs). By dint of such generalization, *Multi-dimensional Dynamic Logic Programming* ($\mathcal{MDLP}$) affords extra expressiveness, thereby enlarging the latitude of logic programming applications unifiable under a single framework. The generality and flexibility provided by a *DAG* ensures a wide scope and variety of new possibilities. By virtue of the newly added characteristics of multiplicity and composition, $\mathcal{MDLP}$ provides a "societal" viewpoint in Logic Programming, important in these web and agent days, for combining knowledge in general. In [10] these new possibilities are explored.

Moreover, an agent not only comprises knowledge about each state, but also knowledge about the transitions between states. The latter can represent the agent's knowledge about the environment's evolution, and its own behaviour and evolution. Since logic programs describe knowledge states, it's only fit that logic programs be utilized to describe transitions of knowledge states as well, by associating with each state a set of transition rules to obtain the next one. In [3], the authors, with others, introduce the language LUPS – "Language for dynamic updates" – designed for specifying changes to logic programs. Given an initial knowledge base (as a logic program) LUPS provides a way for sequentially updating it. The declarative meaning of a sequence of sets of update commands in LUPS is defined by the semantics of the dynamic logic program generated by those commands.

Based on the strengths of $\mathcal{MDLP}$ as a framework capable of simultaneously representing several aspects of a system in a dynamic fashion, and on those of *LUPS* as a powerful language to specify the evolution of such representations via state transitions, we have launched into the design of an agent architecture, $\mathcal{MINERVA}$. Named after the Goddess of Wisdom, this architecture is conceived with the intention of providing, on a sound theoretical basis, a common agent framework based on the strengths of Logic Programming, so as to allow the combination of several non-monotonic knowledge representation and reasoning mechanisms developed in recent years. Rational agents, in our opinion, will require an admixture of any number of those reasoning mechanisms to carrying out their tasks. To this end, a $\mathcal{MINERVA}$ agent hinges on a modular design, whereby a common knowledge base, containing all knowledge shared by more that one sub-agent, is concurrently manipulated by specialized sub-agents.

Its description here is by no means exhaustive, a task that would be impossible to accomplish within this article alone. Rather, it aims at providing an overview highlighting the role of computational logic in its several components.

2 Dynamic Logic Programming

In this section we briefly review Dynamic Logic Programming [1], its generalization $\mathcal{MDLP}$ [9], and the update command language *LUPS* [3].

The idea of dynamic updates is simple and quite fundamental. Suppose that we are given a set of generalized logic program modules P_s (possibly with negation in rule heads), structured by a DAG (or simply index by a sequence number, in [1]). Each program P_s contains some knowledge that is supposed to be true at the state s. Different states may represent different time periods or different sets of priorities or perhaps even different viewpoints. Consequently, the individual program modules may contain mutually contradictory as well as overlapping information. The role of the dynamic program update is to use the mutual relationships existing between different states (and specified by the relation in the DAG) to precisely determine, at any given state s, the *declarative* as well as the *procedural* semantics of the combined program, composed of all modules. The declarative semantics at some state is determined by the stable models of the program that consists in all those rules that are "valid" in that state. Intuitively a rule is "valid" in a state if either it belong to the state or it belong to some previous state in the DAG and is not rejected (i.e. it is inherited by a form of inertia). A rule from a less prioritary state is rejected if there is another conflicting rule in a more prioritary state with a true body. A transformational semantics, that directly provides a means for $\mathcal{MDLP}$ implementation, has also been defined[1]. For details on Dynamic Logic Programming and $\mathcal{MDLP}$ the reader is referred to [1,9]. The paper [10], contains a more detailed review of this subject, as well as an explanation of how $\mathcal{MDLP}$ can be employed to model the combination of inter- and intra-agent societal viewpoints.

LUPS [3] is a logic programming command language for specifying updates. It can be viewed as a language that declaratively specifies how to construct a sequence of logic programs. A sentence *U* in *LUPS* is a set of simultaneous update commands (or actions) that, given a pre-existing sequence of logic programs (or a $\mathcal{MDLP}$), whose semantics corresponds to our knowledge at a given state (or program), produces a new $\mathcal{MDLP}$ with one more program, corresponding to the knowledge that results from the previous sequence after performing all the simultaneous commands. Different possibilities on how to connect the newly produced program with the previously extant ones give rise to different ways of combining the knowledge of the various agents. In [10] these possibilities are examined, and it is shown how to attain: equal role among agents, time prevailing and hierarchy prevailing representations. In this paper, if not otherwise stated, simply assume that the new program is concatenated immediately after the program where the command is evaluated.

A program in *LUPS* is a sequence of such sentences, and its semantics is defined by means of a dynamic logic program generated by the sequence of commands. In [3], a translation of a LUPS program into a generalized logic program is presented, where stable models exactly correspond to the semantics of the original LUPS program. This translation directly provides an implementation of LUPS (available at the URL given above).

[1] The implementation can be found in `centria.di.fct.unl.pt/~jja/updates/`

LUPS update commands specify assertions or retractions to the current program (a predefined one in the $\mathcal{MDLP}$, usually the one resulting from the last update performed). In LUPS a simple assertion is represented as the command:

$$\textbf{assert } L \leftarrow L_1, \ldots, L_k \textbf{ when } L_{k+1}, \ldots, L_m \tag{1}$$

meaning that if $L_{k+1}, \ldots, L_m$ holds in the current program, the rule $L \leftarrow L_1, \ldots, L_k$ is added to the new program (and persists by inertia, until possibly retracted or overridden by some future update command). To represent rules and facts that do not persist by inertia i.e., that are one-state events, LUPS includes the modified form of assertion:

$$\textbf{assert event } L \leftarrow L_1, \ldots, L_k \textbf{ when } L_{k+1}, \ldots, L_m \tag{2}$$

The retraction of rules is performed with the two update commands:

$$\textbf{retract } L \leftarrow L_1, \ldots, L_k \textbf{ when } L_{k+1}, \ldots, L_m \tag{3}$$

$$\textbf{retract event } L \leftarrow L_1, \ldots, L_k \textbf{ when } L_{k+1}, \ldots, L_m \tag{4}$$

meaning that, subject to precondition $L_{k+1}, \ldots, L_m$ (verified at the current program) rule $L \leftarrow L_1, \ldots, L_k$ is either retracted from its successor state onwards, or just temporarily retracted in the successor state (if governed by **event**).

Normally assertions represent newly incoming information. Although its effects remain by inertia (until contravened or retracted), the assert command itself does not persist. However, some update commands may desirably persist in the successive consecutive updates. This is especially the case of laws which, subject to some preconditions, are always valid, or of rules describing the effects of an action. In the former case, the update command must be added to all sets of updates, to guarantee that the rule remains indeed valid. In the latter case, the specification of the effects must be added to all sets of updates, to guarantee that, whenever the action takes place, its effects are enforced. To specify such persistent update commands, *LUPS* introduces:

$$\textbf{always } L \leftarrow L_1, \ldots, L_k \textbf{ when } L_{k+1}, \ldots, L_m \tag{5}$$

$$\textbf{always event } L \leftarrow L_1, \ldots, L_k \textbf{ when } L_{k+1}, \ldots, L_m \tag{6}$$

$$\textbf{cancel } L \leftarrow L_1, \ldots, L_k \textbf{ when } L_{k+1}, \ldots, L_m \tag{7}$$

The first two statements signify that, in addition to any new set of arriving update commands, the persistent update command keeps executing along with them too. The first case without, and the second case with the **event** keyword. The third statement cancels execution of this persistent update, once the conditions for cancellation are met.

3 Overall Architecture

In the $\mathcal{MINERVA}$ agent architecture, an agent consists of several specialized, possibly concurrent, sub-agents performing various tasks while reading and manipulating a common knowledge base. A schematic view of the $\mathcal{MINERVA}$ architecture is depicted in Fig. 1. The common knowledge base contains knowledge about the self and the agent community, and is conceptually divided into the following components: *Capabilities*, *Intentions*, *Goals*, *Plans*, *Reactions*, *Object Knowledge Base* and *Internal Behaviour Rules*. There is also an internal clock. Although conceptually divided in such components, all these modules will share a common representation mechanism based on $\mathcal{MDLP}$ and *LUPS*, the former to represent knowledge at each state and *LUPS* to represent the state transitions i.e., the common part of the agent's behaviour. Every agent is composed of specialized function related sub-agents, that execute their various specialized tasks. Examples of such sub-agents are those implementing the reactive, planning, scheduling, belief revision, goal management, learning, dialogue management, information gathering, preference evaluation, strategy, and diagnosis functionalities. These sub-agents contain a *LUPS* program encoding its behaviour, interfacing with the *Common Knowledge Base*. Whilst some of those sub-agent's functionalities are fully specifiable in *LUPS*, others will require private specialized procedures where *LUPS* serves as an interface language.

In all *LUPS* commands, both in the common knowledge base and in the sub-agents, by default the **when**-clauses are evaluated in the *Object Knowledge Base*, and the rules are added to the *Object Knowledge Base* also. More precisely, the **when**-condition of a sub-agent α_n *LUPS* command is evaluated at a special sink program of the *Object Knowledge Base*[2]. If the condition is verified, the rule is asserted (resp. retracted), by default, at the state $\alpha_{n_{c+1}}$ i.e., the sub-agent's state corresponding to the next time state. In effect, at each time state c, each sub-agent's LUPS interpreter evaluates its commands at state α' (dynamically and therefore differently at each time state) and produces its corresponding next state, $c+1$. Whenever some literal L in a **when**-condition is to be evaluated in a program different from the default one, we denote that by $L@ProgramName$. This @ notation will also be used to denote addition of rules to programs other than the default one.

4 Common Knowledge Base

Object knowledge base. The *Object Knowledge Base* is the main component containing knowledge about the world i.e., knowledge at the object level and info about the society where the agent is situated.

It is represented by an evolving *MDLP*. In it there is a sequence of nodes for each sub-agent of the agent α, representing its evolution in time. There is also a sequence of nodes for every other agent in the multi-agent system, representing α's view of the evolution of those agents in time. As will be seen in the following Section, at each time state each sub-agent manipulates its corresponding state node. There is a *Dialoguer* sub-agent dealing with the interactions with other agents, and manipulating the state nodes corresponding to the agents it communicates with. Several methods are possible for combining these several sequences to form the *MDLP*'s DAG. Each corresponds to a different way of combining inter- and intra-agent social viewpoints. In [10] we show how this can be done.

[2] This special program is a sink of the DAG, and its semantics corresponds to the semantics of all the *Object Knowledge Base*, cf. [10].

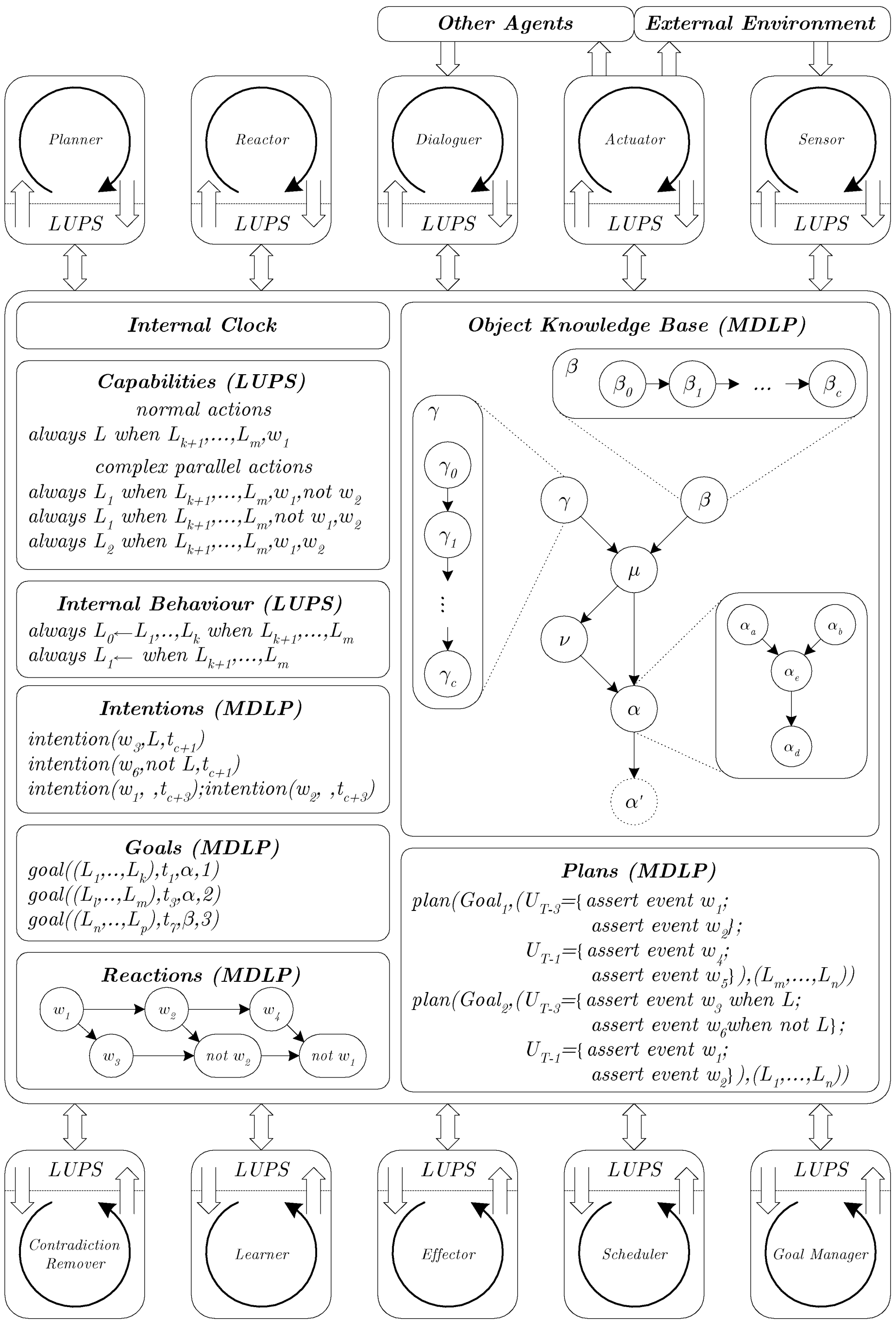

Other Agents
External Environment
Planner
LUPS
Reactor
LUPS
Dialoguer
LUPS
Actuator
LUPS
Sensor
LUPS
Internal Clock
Capabilities (LUPS)
normal actions
always L when L_{k+1},...,L_m,w_1
complex parallel actions
always L_1 when L_{k+1},...,L_m,w_1,not w_2
always L_1 when L_{k+1},...,L_m,not w_1,w_2
always L_2 when L_{k+1},...,L_m,w_1,w_2
Internal Behaviour (LUPS)
always L_0←L_1,..,L_k when L_{k+1},...,L_m
always L_1← when L_{k+1},...,L_m
Intentions (MDLP)
intention(w_3,L,t_{c+1})
intention(w_6,not L,t_{c+1})
intention(w_1, ,t_{c+3});intention(w_2, ,t_{c+3})
Goals (MDLP)
goal((L_1,..,L_k),t_1,α,1)
goal((L_l,..,L_m),t_3,α,2)
goal((L_n,..,L_p),t_7,β,3)
Reactions (MDLP)
w_1
w_2
w_4
w_3
not w_2
not w_1
Object Knowledge Base (MDLP)
β
β_0
β_1
...
β_c
γ
γ_0
γ_1
γ_c
γ
β
μ
ν
α
α'
α_a
α_b
α_e
α_d
Plans (MDLP)
plan(Goal_1,(U_{T-3}={ assert event w_1;
assert event w_2};
U_{T-1}={ assert event w_4;
assert event w_5}),(L_m,...,L_n))
plan(Goal_2,(U_{T-3}={ assert event w_3 when L;
assert event w_6 when not L};
U_{T-1}={ assert event w_1;
assert event w_2}),(L_1,...,L_n))
LUPS
Contradiction Remover
LUPS
Learner
LUPS
Effector
LUPS
Scheduler
LUPS
Goal Manager

Fig. 1. The $\mathcal{MINERVA}$ agent architecture

Capabilities. This block contains a description of the actions, and their effects, capable of being performed by the agent. *LUPS*, by allowing to declaratively specify both knowledge states and their transitions, can be used as a powerful representation language for specifying actions and their effects. Its diversity of update commands can model from the simplest condition-effect action to parallel actions and their coupled indirect effects. Typically, for every *action* ω, there will be one (or more) *LUPS* commands, where the action name ω appears in the '**when**' clause. For example, in

$$\mathbf{always}\ L \leftarrow L_1, \ldots, L_k\ \mathbf{when}\ L_{k+1}, \ldots, L_m, \omega$$

we have, intuitively, that ω is an action whose preconditions are $L_{k+1}, \ldots, L_m$ and whose effect is an update that, according to its type, can model different kinds of actions, all in one unified framework. Examples of kinds of actions are:

- actions of the form "ω **causes** F **if** $F_1, \ldots, F_k$", where $F, F_1, \ldots, F_k$ are fluents (such as in the language $\mathcal{A}$ of [6]) translates into the update command

$$\mathbf{always}\ F\ \mathbf{when}\ F_1, \ldots, F_k, \omega$$

- actions whose epistemic effect is a rule update of the form "ω **updates with** $L \leftarrow L_1, \ldots, L_k$ **if** $L_{k+1}, \ldots, L_m$" translates into the update command

$$\mathbf{always}\ L \leftarrow L_1, \ldots, L_k\ \mathbf{when}\ L, \ldots, L_m, \omega$$

- actions that, when performed in parallel, have different outcomes, of the form "ω_a **or** ω_b **cause** L_1 **if** $L_{k+1}, \ldots, L_m$" and "ω_a **and** ω_b **in parallel cause** L_2 **if** $L_{k+1}, \ldots, L_m$" translate into the three update commands:

$$\begin{gathered}\mathbf{always}\ L_1\ \mathbf{when}\ L_{k+1}, \ldots, L_m, \omega_a, \mathit{not}\, \omega_b \\ \mathbf{always}\ L_1\ \mathbf{when}\ L_{k+1}, \ldots, L_m, \mathit{not}\, \omega_a, \omega_b \\ \mathbf{always}\ L_2\ \mathbf{when}\ L_{k+1}, \ldots, L_m, \omega_a, \omega_b\end{gathered}$$

- actions with non-deterministic effects of the form "α **cause** $L_1 \vee L_2$ **if** $L_{k+1}, ..., L_m$" translates into the update commands (where β_a, β_b are new auxiliary predicates and I is a unique identifier of an occurrence of α):

$$\begin{aligned}\mathbf{always}\ L_1 \leftarrow \beta_a(I)\ \mathbf{when}\ L_{k+1}, \ldots, L_m, \alpha(I) \\ \mathbf{always}\ L_2 \leftarrow \beta_b(I)\ \mathbf{when}\ L_{k+1}, \ldots, L_m, \alpha(I) \\ \mathbf{always}\ \beta_a(I) \leftarrow \mathit{not}\, \beta_b(I)\ \mathbf{when}\ L_{k+1}, \ldots, L_m, \alpha(I) \\ \mathbf{always}\ \beta_b(I) \leftarrow \mathit{not}\, \beta_a(I)\ \mathbf{when}\ L_{k+1}, \ldots, L_m, \alpha(I)\end{aligned}$$

In this representation of the non-deterministic effects of an action α, we create two auxiliary actions (β_a, β_b) with deterministic effects, and make the effect of α be the non-deterministic choice between actions β_a or β_b.

Internal behaviour rules. The *Internal Behaviour Rules* are *LUPS* commands that specify the agent's reactive internal epistemic state transitions. They are of the form:

$$\textbf{assert } L \leftarrow L_1, \ldots, L_k \textbf{ when } L_{k+1}, \ldots, L_m$$

whose effect is the immediate asserting of rule $L \leftarrow L_1, \ldots, L_k$ in the *Object Knowledge Base* if $L_{k+1}, \ldots, L_m$ holds now. These commands will be executed by one of the sub-agents (the *Reactor*), and they are quite common since they may be used by more than one of the sub-agents. E.g. the *Planner* sub-agent must be aware of these rules when planning for goals. An example of internal behaviour rules would be:

$$\begin{array}{l}\textbf{assert } jail(X) \leftarrow abortion(X) \textbf{ when } government(republican)\\ \textbf{assert } not\, jail(X) \leftarrow abortion(X) \textbf{ when } government(democrat)\end{array}$$

stating that whenever the government is republican, an abortion implies going to jail, and whenever the government is democrat, an abortion does not imply going to jail.

Goals. The *Goals* structure is a DLP where at each state the program contains facts of the form $goal\ (Goal, Time, Agent, Priority)$ representing the goals that are to be achieved by the agent. $Goal$ is a conjunction of literals, $Time$ refers to a time state, $Agent$ represents the origin of the goal and $Priority$ contains the priority of the goal. *Goals* is a dynamic structure in the sense that new goals can be asserted onto this structure by any sub-agent, some of them originating in another one. They will be manipulated by a *Goal Manager* sub-agent that can assert, retract and change a goal's priority.

Plans. As shown in [2], *LUPS*, together with abduction, can be used to represent and solve planning problems. There, the notion of *action update*, U^{ω}, is defined as a set of update commands of the form $U^{\omega} = \{\textbf{assert event } \omega\}$ where ω is an action name. Intuitively, each command of the form "**assert event** ω" represents the performing of action ω. Note that performing an action itself is something that does not persist by inertia. Thus, according to the description of *LUPS*, the assertion must be of an event. By asserting **event** ω, the effect of the action will be enforced if the preconditions are met. Each *action update* represents a set of simultaneous actions. Plans are *LUPS* programs composed of action updates. An example of a plan to achieve a $Goal_1$ at time T would be:

$$\begin{array}{l}U_{T-3} = \{\textbf{assert event } \omega_1; \textbf{assert event } \omega_2\}\\ U_{T-1} = \{\textbf{assert event } \omega_4; \textbf{assert event } \omega_5\}\end{array}$$

meaning that in order to achieve $Goal_1$ at time T, the agent must execute actions ω_1 and ω_2 at time $T-3$ and actions ω_4 and ω_5 at time $T-1$. Also, associated with each plan, there is a set of preconditions ($Conditions$) that must be met at the beginning of the plan. Each plan ($Plan$) for goal ($Goal$) is generated by a planner sub-agent, which asserts it into the *Common Knowledge Base* in the form $plan\ (Goal, Plan, Conditions)$ where it is kept for future reuse.

By permiting plans to be represented in the *LUPS* language, we allow the use of more complex planners, namely conditional planners. Conditional action updates are to be represented by *LUPS* commands of the form:

$$\textbf{assert event } \omega \textbf{ when } L_{k+1}, \ldots, L_m$$

An example of such a conditional plan, for $Goal_2$ would be:

$$U_{T-3} = \{\textbf{assert event } \omega_3 \textbf{ when } L; \textbf{assert event } \omega_6 \textbf{ when } not\, L\}$$
$$U_{T-2} = \{\textbf{assert event } \omega_1; \textbf{assert event } \omega_2\}$$

meaning that in order to achieve $Goal_2$ at time T, the agent must execute, at time $T-3$, the command **assert event** ω_3 if L holds at time $T-3$, otherwise, if $not\, L$ holds, it must execute the command **assert event** ω_6. Then, it should execute the commands **assert event** ω_1 and **assert event** ω_2 at time $T-2$. These conditions, in the **when** statement of these commands, can also be employed to include the verification of the success of previous actions.

The plans stored in the common knowledge are going to be processed, together with the reactions, by a specialized *Scheduler* sub-agent that will avail itself of them to determine the intentions of the agent.

Reactions. The *Reactions* block at the *Common Knowledge Base* is represented by a *very simple MDLP* whose object rules are just facts denoting actions ω or negation of actions names $not\, \omega$. The *MDLP* contains a sequence of nodes for every sub-agent capable of reacting, and a set of edges representing a possible hierarchy of reactions. Those sub-agents will contain a set of *LUPS* commands of the form

$$\textbf{assert } \omega \textbf{ when } L_{k+1}, \ldots, L_m$$

Whenever $L_{k+1}, \ldots, L_m$ is true, action ω should be reactively executed. This corresponds to the assertion of ω in the corresponding node of the *Reactions* module. These, together with the plans, will be processed by the *Scheduler* sub-agent to transform them into intentions. The richness of the *LUPS* language enables us to express action blockage i.e., under some conditions, preventing some action ω to be executed by the agent, possibly proposed by another sub-agent. This can be expressed by the command:

$$\textbf{assert } not\, \omega \textbf{ when } L_{k+1}, \ldots, L_m$$

whose effect would be the assertion of $not\, \omega$. This assertion would block any ω proposed by a lower ranked reactive sub-agent. By admitting the buffering of reactions i.e., they are first proposed and then scheduled, this architecture allows for some conceptual control over them, namely by preventing some undesirable reactions being performed.

Intentions. *Intentions* are the actions the agent has committed to. A *Scheduler* sub-agent compiles the plans and reactions to be executed into the intentions, according to the resources of the agent. Control over the level of reactivity and deliberation can be achieved by this process. The *Intentions* structure is a *DLP* where the object language rules are simply facts of the form:

$$intention\,(Action, Conditions, Time)$$

where $Action \in K_\omega$, $Conditions$ is a conjunction of literals from L, and $Time \in T$. They can be interpreted as: execute $Action$, at time T, if $Conditions$ is true. It is left to the Actuator sub-agent to execute the intentions. For example, to achieve $Goal_2$ and $Goal_1$, the two plans previously exemplified can be compiled into the set of intentions (where c is the current time state):

$$\begin{array}{lll} intention\,(\omega_3, L, c+1)\,; & intention\,(\omega_1, \_, c+3)\,; & intention\,(\omega_4, \_, c+5) \\ intention\,(\omega_6, not\,L, c+1)\,; & intention\,(\omega_2, \_, c+3)\,; & intention\,(\omega_5, \_, c+5) \end{array}$$

Although typically intentions are internal commitments, nothing prevents us from having a specialized sub-agent manage (and possibly retract from) the set of intentions.

5 Sub-agents

The sub-agents constitute the labour part of the $\mathcal{MINERVA}$ architecture. Their tasks reside in the evaluation and manipulation of the *Common Knowledge Base* and, in some cases, in the interface with the environment in which the agent is situated. They differ essentially in their specialities, inasmuch as there are planners, reactors, schedulers, learners, dialoguers, etc. Conceiving the architecture based on the notion of, to some extent, independent sub-agents attain a degree of modularity, appropriate in what concerns the adaptability of the architecture to different situations.

Each of these sub-agents contains a *LUPS* program encoding its behaviour, interfacing with the *Common Knowledge Base* and possibly with private specialized procedures. Conceptually, each of these sub-agents also comprises a *LUPS* meta-interpreter that executes its *LUPS* program and produces a sequence of states in the structures of the Common Knowledge Base. The collection of all such sequences of states, produced by all the sub-agents will come to constitute the states of the *Object Knowledge Base*.

To permit private procedure calls, we extend LUPS by allowing those to be called in the **when** statement of a command:

$$\textbf{assert}\ X \leftarrow L_1, \ldots, L_k\ \textbf{when}\ L_{k+1}, \ldots, L_m, ProcCall(L_{m+1}, \ldots, L_n, X)$$

In this case, if $L_{k+1}, \ldots, L_m$ is true at state α', then $ProcCall(L_{m+1}, \ldots, L_n, X)$ is executed and, if successful, X is instantiated and rule $X \leftarrow L_1, \ldots, L_k$ is asserted into the sub-agent's state corresponding to the next time state. These procedure calls can read from the *Common Knowledge Base*, but cannot change it. All changes to the Common Knowledge Base are performed via *LUPS* commands, ensuring a clear and precise semantics of the overall system.

It is important to stress that *LUPS* constitutes a language to specify state transitions, namely by determining the construction of the programs that define each state, whose meaning is assigned by the semantics of $\mathcal{MDLP}$. Some of the tasks to be performed by the sub-agents need to resort to procedure calls, but others are fully specifiable with *LUPS* commands that simply consult the Common Knowledge Base.

Sensor. The *Sensor* sub-agent receives input from the environment through procedure calls of the form $SensorProc(Rule)$. The information, in the form of logic program rules, is asserted in the object knowledge base. This behaviour is accomplished with *LUPS* commands of the form:

$$\textbf{assert}\ Rule\ \textbf{when}\ SensorProc(Rule)$$

The **when** statement can also be used as a filter, possibly specifying conditions for the acceptance of the input. Mark that since there is no explicit reference as where to execute the command, the assertions are, by default, performed at the *Sensor*'s node at the *Object Knowledge Base*.

Dialoguer. The Dialoguer sub-agent is similar to the *Sensor* one in what concerns receiving input from the environment. It differs in that it processes incoming messages from the other agents in the multi-agent system ($\beta, \gamma, \mu, ...$), and the *LUPS* commands are then executed by asserting the information into the corresponding agent's node. Moreover, it is this sub-agent's charge, according to the message, to generate new goals, issue replies, etc. Examples of *LUPS* commands typical of this sub-agent would be:

$$\begin{array}{l}
\mathbf{assert}\ goal(Goal, Time, Agent, \_)@goals \\
\qquad\qquad \mathbf{when}\ MsgFrom(Agent, Goal, Time, Rule), cooperative(Agent). \\
\mathbf{assert}\ Rule@Agent\ \mathbf{when}\ MsgFrom(Agent, Goal, Time, Rule). \\
\mathbf{assert}\ msgTo(Agent, Goal, plan(Goal, Plan, Cond))@reactions \\
\qquad\qquad \mathbf{when}\ goal(Goal, Time, Agent, \_)@goals, Agent \neq \alpha, \\
\qquad\qquad\qquad plan(Goal, Plan, Cond)@plans.
\end{array}$$

These commands specify a behaviour for the acceptance of goals and beliefs from another agent, and for dealing with the request. The first command insists that a goal should be asserted into the *Goals* of the *Common Knowledge Base* whenever a message from a cooperative agent requests it. The second asserts the rules communicated by another agent into its corresponding node of the *Object Knowledge Base*. The third issues a communication action, by asserting it into the *Reactions*, to be scheduled, whenever there is a request from another agent to solve a goal, and there is a plan for such a goal.

Actuator. The Actuator sub-agent interfaces with the environment by executing actions on it. At each cycle, the *Actuator* extracts the actions to be executed from the *Intentions* and performs them on the outside world. If successful, it event asserts the action name, ω, in the *Object Knowledge Base*. At each cycle, the *Actuator* executes LUPS commands of the form:

$$\begin{array}{l}
\mathbf{assert\ event}\ \omega\ \mathbf{when}\ intention\ (Action, Cond, Time)\ @intentions, \\
\qquad\qquad\qquad Current(Time), Cond, ActionProc(\omega)
\end{array}$$

corresponding to the execution of the *Intentions* scheduled for the current time state.

Effector. The Effector, at each cycle, evaluates and executes the LUPS commands in the Capabilities and Common behaviour Rules. Note that, although the *Capabilities* and *Common Behaviour Rules* constitute the *LUPS* program of the *Effector* sub-agent, they belong to the *Common Knowledge Base* because they may be accessed by other sub-agents, namely by the Planner. The *Capabilities* require the prior successful execution of an action. This does not happen with the *Common Behaviour Rules*. These specify state transitions solely based on the internal state of the agent.

Reactor. The *Reactor* will contain reactive rules that, if executed, produce an action to be performed. They are of the form:

$$\textbf{assert event } \omega @ \mathit{reactions} \textbf{ when } L_1, \ldots, L_k$$

An example of such a reaction would be

$$\textbf{assert event } \mathit{runaway} @ \mathit{reactions} \textbf{ when } \mathit{danger}$$

where $\mathit{runaway}$ is the name of an action. We can also reactively and temporarily block actions by using *LUPS* commands of the form:

$$\textbf{assert event } \mathit{not}\, \omega @ \mathit{reactions} \textbf{ when } L_1, \ldots, L_k$$

Planner. In [2], it was shown how planning can be achieved by means of abduction in LUPS specified scenarios. Such an abduction based planner uses the *Object Knowledge Base* together with the *Intentions* (actions already set to be executed in the future), *Capabilities*, and the *Common Behaviour Rules*, to achieve a plan for a given goal, consistent with the current set of intentions. A *LUPS* command for the *Planner*, with such an abductive planner represented by the procedure call $\mathit{AbductPlan}(\mathit{Goal}, \mathit{Plan}, \mathit{Cond})$, would be:

$$\begin{array}{l} \textbf{assert } \mathit{plan}\,(\mathit{Goal}, \mathit{Plan}, \mathit{Cond})\, @\mathit{plans} \\ \quad \textbf{when } \mathit{goal}\,(\mathit{Goal}, T, \_, 1)\, @\mathit{goals},\ \mathit{AbductPlan}(\mathit{Goal}, \mathit{Plan}, \mathit{Cond}) \end{array}$$

Other planners, possibly more efficient and/or complex can be used, namely partial planners, with the appropriate *LUPS* commands to interface them.

Goal manager. Goals can be asserted by other sub-agents, some of them originating in other agents. These goals may conflict with one another. The task of the *Goal Manager* is to deal with such issues. It manipulates the *Goals* structure, being able to merge goals, set priorities, delete goals, etc. An example of a command for this sub-agent, specifying that if two goals are incompatible, the one with lower priority should be removed, is:

$$\begin{array}{ll} \textbf{retract } \mathit{goal}\,(G_1, T_1, A_1, P_1)\, @\mathit{goals} & \\ \quad \textbf{when } & \mathit{goal}\,(G_1, T_1, A_1, P_1)\, @\mathit{goals}, \mathit{goal}\,(G_2, T_2, A_2, P_2)\, @\mathit{goals}, \\ & \mathit{incomp}(G_1, G_2), P_1 < P_2 \end{array}$$

Scheduler. The *Scheduler* determines the Intentions of the agent, based on the current state of the agent. The Scheduler acts whenever there are pending reactions or goals and plans. More than one specialized scheduling procedure may exist. For example, if there were three procedures, depending on the need to combine reactions and plans or just to schedule one of them, the following LUPS commands would select their execution:

$$\begin{array}{ll} \textbf{assert } \mathit{\Pi} @\mathit{intentions} \textbf{ when} & \mathit{goal}\,(G, \_, \_, \_)\, @\mathit{goals}, \mathit{plan}(G, \_, \_) @\mathit{plans}, \\ & \mathit{not}\, X @\mathit{Reactions}, \mathit{SchedulePlans}(\mathit{\Pi}). \\ \textbf{assert } \mathit{\Pi} @\mathit{intentions} \textbf{ when} & \mathit{not}\, \mathit{goal}\,(\_, \_, \_, \_)\, @\mathit{goals}, X @\mathit{reactions}, \\ & \mathit{ScheduleReactions}(\mathit{\Pi}). \\ \textbf{assert } \mathit{\Pi} @\mathit{intentions} \textbf{ when} & \mathit{goal}\,(G, \_, \_, \_)\, @\mathit{goals}, \mathit{plan}(G, \_, \_) @\mathit{plans}, \\ & X @\mathit{reactions}, \mathit{CombineSchedule}(\mathit{\Pi}). \end{array}$$

Other sub-agents. Other sub-agents can concurrently exist in the $\mathcal{MINERVA}$ architecture. Fig. 1 mentions a *Contradiction Remover* and a *Learner* sub-agent, but many others can be specified. Either fully specified by means of *LUPS* commands or with *LUPS* serving simply as an interface to procedure calls or legacy code, a great variety of sub-agents can be fully integrated into this architecture. If tight control is needed, *LUPS* also encompasses the specification of control sub-agents. For examples of the application of *LUPS* in several domains the reader is referred to [4].

6 Related Work

The characteristic of $\mathcal{MINERVA}$ that makes it most distinct from other existing agent architectures is that its knowledge is represented by non-monotonic theories, expressed by logic programs, with default negation in both the heads and bodies of clauses, organized according to acyclic digraphs. This permits the use of the richer knowledge representation capabilities afforded by such logic programs, for example to represent the relations amongst the agents themselves in a MAS, as well as for a direct and simple mechanism to represent and reason about dynamic environments where the governing rules change with time. It has been shown [1] that updating non-monotonic theories is quite different from updating theories represented in classical propositional logic, governed by the update postulates in [8], and far different from the mere assertion and deletion of formulas. To the best of our knowledge, $\mathcal{MINERVA}$ is the only agent architecture that proposes to incorporate such update capabilities. Most other agent architectures incorporate some update facilities, but all in the line of the update postulates of [8], or by simple assertion and retraction of formulas. We should also make clear at this point that the assertions and retractions specified by the *LUPS* commands differ substantially from those approaches employing simple assertion and retraction of formulas. The former are dealt with by the semantics of $\mathcal{MDLP}$, which further rejects other existing rules on the grounds of the newly asserted ones and may as well reinstate other previously rejected rules, while the latter only deal with "physical" assertions and retractions of formulas, the semantics being determined according to the new set of formulas.

Clearly, not all situations require such update features as our's and, in those situations, other existing agent proposals may perform better. Nevertheless, when the governing rules change dynamically, new rules possibly invalidating existing rules, as for example in legal reasoning scenarios, a semantics such as the one provided for $\mathcal{MDLP}$ will be required. $\mathcal{MDLP}$ additionally caters for a clear framework, with a precise semantics, to represent societies of hierarchically related agents.

In [7], Hindriks et al. propose the agent programming language *3APL*, together with a transition system based operational semantics. Knowledge is represented as a set of first order formulas (although the authors claim it could be any logical language) in a database and goals are sequences of imperative and logic constructs. On top of these there are the so-called practical reasoning rules to specify changes to the agent state. *3APL*, as claimed by the authors, intends to be a language to serve both as a sound basis for theoretical work on agents and as a vehicle for practical applications. The approach to the design of *3APL* is based on the premise that an agent programming language amounts to providing a way to specify the dynamics of the mental state of an agent, be it its knowledge and beliefs or its goals, intentions, and commitments. This intention and premise are shared by $\mathcal{MINERVA}$ in the sense that the first is the ground for the choice of non-monotonic logic programming in general as the basic framework, and the second amounts to the motivation for using multi-dimensional dynamic logic programming and *LUPS* as knowledge representation and evolution specification language. In some sense, we can compare the *3APL* database to the object knowledge base of a $\mathcal{MINERVA}$ agent and the *3APL* practical reasoning rules to the *LUPS* commands in $\mathcal{MINERVA}$. The differences being that, in $\mathcal{MINERVA}$, updates to the environment (and to the agent mental state) that can bring about contradictory theories (if formulas are simply regarded as all having the same strength) are automatically dealt with by *LUPS* and $\mathcal{MDLP}$. As noted above, updates are not simple assertions and retractions of formulas. In particular, a newly asserted rule is not equal in "power" to an old rule (in that case it would be a revision and not an update) and does not simply replace it. Katsuno and Mendelzon [8] study the updates for theories in classical propositional logic. Updating a belief base depends on the representation language. In particular, updates of non-monotonic knowledge bases, for example represented by logic programs under the stable models semantics, require a substantially different mechanism from those of classical propositional logic based representations, as discussed in [1]. It seems that, unlike the authors claim, the belief base used by *3APL* agents cannot be represented by an arbitrary logic language. In particular, their belief base cannot use the representational capabilities of default negation. $\mathcal{MINERVA}$ has been specifically designed to use logic programs with default negation both in the heads and in the bodies of clauses as the basic representation mechanism. Programs written in such a language are arranged according to several acyclic digraphs whose semantics is precisely characterized by $\mathcal{MDLP}$. The use of *LUPS*, which has an imperative reading, provides for the mental state transitions by means of the necessary update operations. In *3APL* there is no notion of sub-agents concurrently changing the mental state of the agent with possibly contradictory information. In $\mathcal{MINERVA}$, these contradictions are solved by the semantics of $\mathcal{MDLP}$ according to a specified hierarchy between those sub-agents. Like in *3APL*, action specifications in *LUPS* are static inasmuch as *LUPS* commands cannot be updated. We are currently extending *LUPS* programs with self-update capabilities to overcome this drawback, and to enrich the behaviour specification power of *LUPS*.

MetateM (and *Concurrent MetateM*), introduced by Fisher [5], is a programming language based on the notion of direct execution of temporal formulae, primarily used to specify reactive behaviours of agents. It shares similarities with the *LUPS* language inasmuch as both employ rules to represent a relation between the past and the future i.e., each rule in *MetateM* and in *LUPS* consists of conditions about the past (or present) and a conclusion about the future. While the use of temporal logics in MetateM allows for the specification of rather elaborate temporal conditions, something for which *LUPS* was not designed for, the underlying $\mathcal{MDLP}$ semantics of *LUPS* allows the specification of agents capable of being deployed in dynamic environments where the governing laws change over time. If we move to the more general class of logic programs with non-monotonic default negation both in the premises and conclusions of clauses, *LUPS* and $\mathcal{MDLP}$ directly provide an update semantics needed to resolve contradictions naturally arising from conflicting rules acquired at different time instants, something apparently not possible in *MetateM*. This partially amounts to the distinction between updating theories represented in classical logic and those represented by non-monotonic logic programs (cf. [1]).

The *IMPACT* agent architecture, introduced by Subrahmanian et al. [15], provides a framework to build agents on top of heterogeneous sources of knowledge. To "agentize" such sources, the authors introduce the notion of agent program written over a language of so-called code-calls, which can be seen as encapsulations of whatever the source represents, and the actions the agent can execute. Such agent programs and their semantics resemble logic programs extended with deontic modalities. In *IMPACT*, at every state transition, the agent determines a set of actions to be executed, obeying some notion of deontic consistency, which simultaneously corresponds to a basic change in the agent's mental state. *IMPACT* agents do not incorporate the rule based update mechanisms of $\mathcal{MINERVA}$ although, in principle, it appears to be possible to specify a set of code-calls for the object knowledge base of a $\mathcal{MINERVA}$ agent, accessing an $\mathcal{MDLP}$ meta-interpreter, and defining the actions (corresponding to the *LUPS* commands) that change the MDLP. In other words, one could agentize a $\mathcal{MINERVA}$ agent to become an *IMPACT* agent.

7 Conclusions

We began this article by extolling the virtue and the promise of Computational Logic, most adeptly in the guise of Logic Programming, for adumbrating the issues of, and the solutions for, an integrated multi-agent architecture with a precisely defined declarative semantics. As we near the end of our proctrated exposition, we are confident to have brought the point home. To wit, our architectural scaffolding rests on a sound and wide basis, whose core are the evolving updatable agents and their attending, and multi-dimensionally configured, knowledge bases.

The basic architecture affords us too, we purport to have shown, with the elasticity and resilience to further support a spate of crucial ancillary functionalities, in the form of additional specialized agents, via compositional, communication, and procedural mechanisms. The circum-exploration of the territories under purview has hardly started, but the means of locomotion appear to be already with us.

Logic Programming is, without a doubt, the privileged melting pot for the articulate integration of functionalities and techniques, pertaining to the design and mechanization of complex systems, addressing ever more demanding and sophisticated computational abilities. For instance, consider again those reasoning abilities mentioned in previous sections. Forthcoming rational agents, to be realistic, will require an admixture of any number of them to carrying out their tasks. No other computational paradigm affords us with the wherewithal for their coherent conceptual integration. And, all the while, the very vehicle that enables testing its specification, when not outright its very implementation.

Acknowledgements

We would like to thank the POCTI project FLUX. João Leite is partially supported by PRAXIS XXI scholarship no. BD/13514/97. Thanks are due to the anonymous referees for their remarks which helped us in improving the paper.

References

1. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming*, 45(1-3):43–70, 2000. A short version titled *Dynamic Logic Programming* appeared in A. Cohn and L. Schubert (eds.), *KR'98*, Morgan Kaufmann.
2. J. J. Alferes, J. A. Leite, L. M. Pereira, and P. Quaresma. Planning as abductive updating. In D. Kitchin, editor, *Proceedings of the AISB'00 Symposium on AI Planning and Intelligent Agents*, pages 1–8. AISB, 2000.
3. J. J. Alferes, L. M. Pereira, H. Przymusinska, and T. Przymusinski. LUPS : A language for updating logic programs. *Artificial Intelligence*, 2001. To appear. A short version appeared in M. Gelfond, N. Leone and G. Pfeifer (eds.), LPNMR-99, LNAI 1730, Springer.
4. J. J. Alferes, L. M. Pereira, T. Przymusinski, H. Przymusinska, and P. Quaresma. An exercise with dynamic logic programming. In L. Garcia and M. Chiara Meo, editors, *Proceedings of the 2000 Joint Conference on Declarative Programming (AGP-00)*, 2000.
5. M. Fisher. A survey of concurrent METATEM: The language and its applications. In Dov M. Gabbay and Hans Jürgen Ohlbach, editors, *Proceedings of the 1st International Conference on Temporal Logic*, volume 827 of *LNAI*, pages 480–505, Berlin, 1994. Springer.
6. M. Gelfond and V. Lifschitz. Action languages. *Linkoping Electronic Articles in Computer and information Science*, 3(16), 1998.
7. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J. C. Meyer. Formal semantics for an abstract agent programming language. In Munindar P. Singh, Anand Rao, and Michael J. Wooldridge, editors, *Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, volume 1365 of *LNAI*, pages 215–230, Berlin, 1998. Springer.
8. Hirojumi Katsuno and Alberto O. Mendelzon. On the difference between updating a knowledge base and revising it. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 387–394, San Mateo, CA, USA, 1991. Morgan Kaufmann Publishers.
9. J. A. Leite, J. J. Alferes, and L. M. Pereira. Multi-dimensional dynamic knowledge representation. In Thomas Eiter, Wolfgang Faber, and Mirosaw Truszczynski, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, volume 2173 of *LNAI*, pages 365–378. Springer, 2001. A preliminary version appeared in F. Sadri and K. Satoh (Eds.) Procs. of CLIMA'00.
10. J. A. Leite, J. J. Alferes, and L. M. Pereira. On the use of multi-dimensional dynamic logic programming to represent societal agents' viewpoints. In P. Brazdil and A. Jorge, editors, *Progress in Artificial Intelligence, 10th Portuguese International Conference on Artificial Intelligence (EPIA-01)*, volume 2258 of *LNAI*. Springer, 2001.
11. I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In J. Dix, U. Furbach, and A. Nerode, editors, *LPNMR-97*, volume 1265 of *LNAI*, pages 420–429, Berlin, 1997. Springer.
12. S. Rochefort, F. Sadri, and F. Toni, editors. *Proceedings of the International Workshop on Multi-Agent Systems in Logic Programming*, Las Cruces, New Mexico, USA, 1999. Available from `http://www.cs.sfu.ca/conf/MAS99`.

13. F. Sadri and F. Toni. Computational logic and multiagent systems: A roadmap, 1999. Available from `http://www.compulog.org`.
14. D. De Schreye, M. Hermenegildo, and L. M. Pereira. Paving the roadmaps: Enabling and integration technologies, 2000. Available from `http://www.compulog.org/net/Forum/Supportdocs.html`.
15. V. S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert Ross. *Heterogeneous Agent Systems*. MIT Press/AAAI Press, Cambridge, MA, USA, 2000.
16. XSB-Prolog. The XSB logic programming system, version 2.0, 1999. Available at `http://www.cs.sunysb.edu/ sbprolog`.

Running AgentSpeak(L) Agents on SIM_AGENT*

Rodrigo Machado and Rafael H. Bordini

Informatics Institute, Federal University of Rio Grande do Sul
Porto Alegre RS 91501-970, Brazil
{rma,bordini}@inf.ufrgs.br

Abstract. This paper presents what is, to the best of our knowledge, the very first successful attempt at running AgentSpeak(L) programs. AgentSpeak(L)is a programming language for BDI agents, created by Rao, with which he pointed for the first time towards bridging the gap between BDI logics and implemented BDI systems. Moreover, it has quite an elegant and neat notation for a BDI programming language, which could establish a turning point in the practice of implementing cognitive multi-agent systems, should it be turned into a working interpreter or compiler. Precisely because such (implemented) interpreter or compiler was unavailable, AgentSpeak(L)has been neglected, as have other agent-oriented programming languages with a strong theoretical support, by multi-agent system practitioners. This paper shows a way of turning AgentSpeak(L)agents into running programs within Sloman's SIM_AGENT toolkit. We have called this prototype interpreter SIM_Speak, and we have tested it with a multi-agent traffic simulation. We also discuss the limitations and possible extensions to SIM_Speak.

1 Introduction

The AgentSpeak(L) programming language was introduced by Rao in [15]. In that paper, not only has he defined formally the operation of an abstract interpreter for it, but he has also started the definition of a proof system for that language with which it was possible to prove, so he claimed, known properties that are satisfied by BDI systems, which had been previously proved with BDI logics [16,18]. Further, he claimed that there is an one-to-one correspondence between his interpreter and the proof system (which is in fact quite similar to the usual inference rules defining the transition relation on configurations used for giving structural operational semantics [14] to programming languages). In this way, he proposed what can be considered as the first viable approach to bridging the ever so quoted gap between BDI theory and practice: an old worry of the BDI community.

Further formalisation of the abstract interpreter and missing details were given by d'Inverno and Luck in [5], using the Z formal specification language [22]. Many of the ideas in that formalisation had already appeared in the formalisation of dMARS in Z, found in [4]. This highlights the fact that AgentSpeak(L) is strongly based on the experience with Kinny's dMARS [11] and Georgeff's PRS [7]. In [5], d'Inverno and Luck claim that their Z specifications can also be used as a framework for the formal specification of other BDI systems. They also pointed out (as future work) to

* This work has been supported by FAPERGS (research grants 99/51044-3 and 99/00358-7) and CNPq (research grants 300409/99-0 and 381489/00-0).

the possibility of animating their Z specifications. Although this has not as yet been done, it is being attempted (see [6]). In effect, our previous experience with animating Z specifications has shown that this is not always as straightforward a job as one would have expected (cf. [2]).

AgentSpeak(L) has many similarities with traditional logic programming, which is another characteristic that would favour its becoming a popular language: it should prove quite intuitive for those familiar with logic programming. Besides, it has a neat notation and provides quite elegant specifications of BDI agents. This could contribute for its being considered also for the production of formal specification of BDI systems, should the theoretical work started by Rao be fully carried out. The fact that Rao's abstract interpreter had not yet been actually implemented accounts for the misfortune of AgentSpeak(L) being completely dismissed by the BDI practitioners community. Likewise, the lack of further theoretical work on Rao's proof system possibly accounts for its being not used even for formal specification of BDI systems. We here aim at giving an initial solution to the practical side of the problem, while the theoretical work is being consider in other research projects in which we are engaged.

Our recent experience with using AgentSpeak(L) as specification language for BDI agent applications [17] has led us to even further detailing of the language and its interpreter, as it happened with d'Inverno and Luck's work. Also, we are developing a purpose-built interpreter for AgentSpeak(L) programs. However, in order to be able to run our AgentSpeak(L) applications before that complex implementation work is done, we have found the means to running AgentSpeak(L) programs within Sloman's SIM_AGENT toolkit [21,20]. This paper presents precisely the mechanism we have created for the conversion of AgentSpeak(L) programs into running code within SIM_AGENT; we call this prototype interpreter SIM_Speak. Besides clarifying issues of the language and its interpreter, this work also produces, peripherally, an interesting comparison between AgentSpeak(L) and SIM_AGENT, two rather different approaches to "cognitively rich" agent systems.

We have been using SIM_AGENT in a project related to Social Simulation (which is mentioned in passing towards the end of the paper). It is a very robust system, based on POP-11 [1] and several libraries developed for it. Because of this previous experience with SIM_AGENT, we have chosen it for this implementation. However, as we mention later, it has proved quite handy in such task. The main advantage of a purpose-built interpreter as opposed to running AgentSpeak(L) programs on SIM_AGENT is arguably in terms of performance. Although for some applications (e.g., those that have real-time requirements) performance is an important issue, this SIM_Speak platform may be useful quite generally.

This paper is structured as follows. The next two sections provide the necessary background on both AgentSpeak(L) and SIM_AGENT. Section 4 shows our solution to the problem of running AgentSpeak(L) programs on SIM_AGENT, and Section 5 gives further details on current implementation, future extensions, and discussion on related work. In Section 6, we discuss our on going projects related either to the implementation of AgentSpeak(L) or our use of the SIM_AGENTtoolkit.

2 AgentSpeak(L)

This section covers the basics of the syntax and informal semantics of AgentSpeak(L) as given in [15], the paper which introduced it. Although in this section the formal definitions of AgentSpeak(L) syntax (Definitions 1 to 8) are reproduced almost like the ones given in that paper (except that we have improved some of them here), this presentation is indispensable, together with the presentation of SIM_AGENT in the next section, for the understanding of the remainder of the paper. As we mentioned before, besides [15], the reader can refer to [5] for detailed formalisation of the language.

2.1 Syntax

An AgentSpeak(L) agent is created by the specification of a set of base beliefs and a set of plans. The definitions below introduce the necessary notions for the specifications of such sets. Those familiar with Prolog, when reading actual examples of AgentSpeak(L) programs (e.g. in [15] and [17]), will notice many similarities, including the convention of using uppercase initials for variable identifiers, and the issues related to predicates, terms, variables, and unification.

Definition 1 (Belief). *If b is a predicate symbol, and $t_1, \ldots, t_n$ are terms then $b(t_1, \ldots, t_n)$ or $b(\mathbf{t})$ is a* belief atom. *A ground belief atom is a* base belief. *A belief atom $b(\mathbf{t})$ or its negation $\neg b(\mathbf{t})$ are* belief literals. *Belief literals are* beliefs. *If φ and ψ are beliefs, $\varphi \wedge \psi$ is also a belief.*

AgentSpeak(L) distinguishes two types of goals: *achievement goals* and *test goals*. Achievement goals are predicates (as defined for beliefs above) prefixed with the '!' operator, while test goals are prefixed with the '?' operator. Achievement goals state that the agent wants to achieve a state of the world where the associated predicate is true. Test goals state that the agent wants to test whether the associated predicate is a true belief (i.e., whether it can be unified with that agent's base beliefs).

Definition 2 (Goal). *If g is a predicate symbol, and $t_1, \ldots, t_n$ are terms then $!g(t_1, \ldots, t_n)$ or $!g(\mathbf{t})$ and $?g(t_1, \ldots, t_n)$ or $?g(\mathbf{t})$ are* goals.

Next, the notion of triggering event is introduced. It is a very important concept in this language, as it is a triggering event that starts off the execution of plans. There are two types of triggering events: those related to the *addition* ('$+$') and those related to the *deletion* ('$-$') of mental attitudes (beliefs or goals, in fact).

Definition 3 (Triggering Event). *If $b(\mathbf{t})$ is a belief atom, $!g(\mathbf{t})$ and $?g(\mathbf{t})$ are goals, then $+b(\mathbf{t})$, $-b(\mathbf{t})$, $+!g(\mathbf{t})$, $-!g(\mathbf{t})$, $+?g(\mathbf{t})$, and $-?g(\mathbf{t})$, are* triggering events.

Clearly, from the usual model of agents (see, e.g., the diagram used in [24]), in regard to their acting on a environment, one sees that plans need to refer to the basic actions that an agent is able to perform on its environment. Such actions are defined below.

Definition 4 (Action). *If a is an action symbol and $t_1, \ldots, t_n$ are first-order terms, then $a(t_1, \ldots, t_n)$ or $a(\mathbf{t})$ is an* action.

We now round off the presentation of AgentSpeak(L) syntax with the definition of plans. Recall that the designer of an agent using AgentSpeak(L) does so by specifying a set of base beliefs and a set of plans only. An AgentSpeak(L) plan has a head which is formed of a triggering event (the purpose for that plan), and a conjunction of belief literals forming a context that needs be satisfied if the plan is to be executed (the context must be a logical consequence of that agent's set of base beliefs). A plan has also a body, which is a sequence of basic actions or (sub) goals that the agent has to achieve (or test).

Definition 5 (Plan). *If e is a triggering event, $b_1, \ldots, b_m$ are belief literals, and $h_1, \ldots, h_n$ are goals or actions, then $e : b_1 \wedge \ldots \wedge b_m \leftarrow h_1; \ldots; h_n$ is a* plan. *The expression to the left of the arrow is referred to as the* head *of the plan and the expression to the right of the arrow is referred to as the* body *of the plan. The expression to the right of the colon in the head of a plan is referred to as the* context. *For convenience, we shall rewrite an empty body with the expression "$true$".*

We now turn to providing the basics on the interpretation of AgentSpeak(L) programs. For that, we need a few more definitions.

Intentions are particular courses of actions (in the form of a stack of partially instantiated plans) to which an agent has committed in order to achieve a particular goal, and are represented as defined below.

Definition 6 (Intention). *Each* intention *is as stack of* partially instantiated plans*, i.e., plans where some of the variables have been instantiated. An intention is denoted by $[p_1 \ddagger \ldots \ddagger p_z]$, where p_1 is the bottom of the stack and p_z is the top of the stack. The elements of the stack are delimited by $\ddagger$. For convenience, we shall refer to the intention* `[+!true:true <- true]` *as the* true intention.

Events, which may start off the execution of plans, can be external, when originating from perception of the agent's environment, or internal, when generated from the agent's own execution of a plan (e.g., a subgoal in the body of a plan is an addition of goal which may be a triggering event). In the latter case, the event is accompanied by the intention which generated it.

Definition 7 (Event). *Each* event *is a tuple $\langle e, i \rangle$, where e is a triggering event and i is an intention. If the intention i is the true intention, the event is called an* external event*; otherwise it is an* internal event.

It is now possible to formally define an AgentSpeak(L) agent as follows.

Definition 8 (Agent). *An* agent *is given by a tuple $\langle E, B, P, I, A, \mathcal{S}_\mathcal{E}, \mathcal{S}_\mathcal{O}, \mathcal{S}_\mathcal{I} \rangle$, where E is a set of events, B is a set of base beliefs, P is a set of plans, I is a set of intentions, and A is a set of actions. The selection function $\mathcal{S}_\mathcal{E}$ selects an event from the set E; the selection function $\mathcal{S}_\mathcal{O}$ selects an option or an applicable plan from a set of applicable plans; and $\mathcal{S}_\mathcal{I}$ selects an intention from the set I.*

2.2 Informal Semantics

The operation of an abstract interpreter for AgentSpeak(L) has been formalised in [15] and [5]. As we do not have the space to present any formal account of an AgentSpeak(L) interpreter here, we have devised a diagram which explains informally the functioning of an interpreter for AgentSpeak(L). Our pictorial description of such interpreter, given in Figure 1, greatly facilitates the understanding of the abstract interpreter proposed by Rao. In the figure, sets (of beliefs, events, plans, and intentions) are represented as rectangles, while diamonds represent selection (of one element from a set), and circles represent some of the processing involved in the interpretation of AgentSpeak(L) programs. The numbers in the circles and diamonds give the order in which the parts of the abstract interpreter are executed in every interpretation cycle.

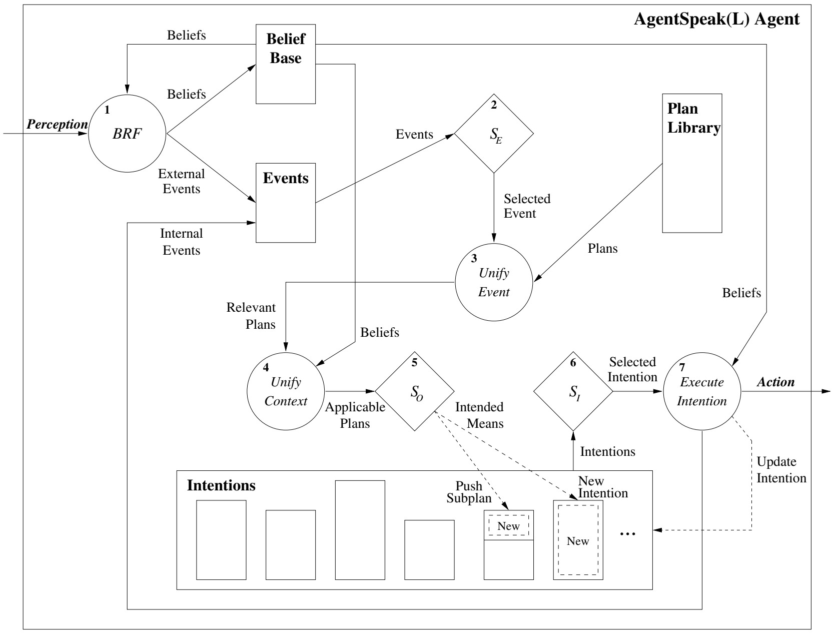


Fig. 1. Interpreting AgentSpeak(L) Programs

At every interpretation cycle of an agent program, AgentSpeak(L) updates a list of events, which may be generated from perception of the environment, or from the execution of intentions (when subgoals are specified in the body of plan). Note that we have introduced a Belief Revision Function (BRF) in the architecture which is implicit in Rao's interpreter (but normally made explicit in the generic BDI architecture [23]). Rao assumes that beliefs are updated from perception and whenever there are changes in the agent's beliefs, this implies the insertion of an event in the set of events. We have made this process explicit in the figure, by including the BRF component.

It is important to remember that $\mathcal{S}_\mathcal{E}$, $\mathcal{S}_\mathcal{O}$, and $\mathcal{S}_\mathcal{I}$ are part of the definition of an agent (see Definition 8). Neither Rao nor d'Inverno and Luck elaborate on how users specify such functions, but they are assumed to be agent-specific. After $\mathcal{S}_\mathcal{E}$ has selected an event, AgentSpeak(L) has to unify that event with triggering events in the heads of plans. This generates a set of all *relevant plans*. When unifying the context part of heads of plans in that set with the agent's beliefs, AgentSpeak(L) determines a set of *applicable plans* (plans that can actually be used for handling the chosen event). Then $\mathcal{S}_{\mathcal{O}}$ chooses a single applicable plan from that set, called the *intended means*, and either pushes that plan on the top of an existing intention (if the event was an internal one), or creates a new intention in the set of intentions (if the event was external, i.e., generated from perception of the environment). Each of an agent's intentions is, therefore, a stack of partially instantiated plans.

All that remains to be done at this stage is to select a single intention to be executed in that cycle. Note that each external event for which there is an applicable plan generates an separate stack of partially instantiated plans within the set of intentions. The $\mathcal{S}_{\mathcal{I}}$ function selects one of the agent's intention (i.e., one of the independent stacks of plans within the set of intentions). On the top of that intention there is a plan, and the formula in the beginning of its body is taken for execution. This implies that either a basic action is performed by the agent on its environment, a subgoal generates an internal event (in case it is an achievement goal), or a test goal is performed (which means that the set of beliefs need be consulted). If the formula being executed is a basic action or a test goal, the set of intention needs be updated. In the case of test goals, further variable instantiation will occur in the partially instantiated plan which contained that test goal (and the test goal itself is removed from the intention from which it was taken). In the case where a basic action is selected, the necessary updating of the set of intentions is simply to remove that action from the intention. When a removed formula marks the end of the body of a subplan, the subgoal that generated it (which therefore stays in the beginning of the body of the plan immediately below it in the stack) is also removed from the intention, or the whole intention is removed from the set if the initial plan (i.e., the plan triggered by an external event) is the one that finished execution. This ends a cycle of execution, and AgentSpeak(L) starts all over again, checking the state of the environment after agents have acted on it, generating events, and so forth.

3 SIM_AGENT

Sloman's SIM_AGENT toolkit was presented in, e.g., [21,20] and is available for download at URL `http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html`. This section introduces the main aspects of SIM_AGENT so that the reader may understand the next section, where we present a way of running AgentSpeak(L) programs on SIM_AGENT.

SIM_AGENT was developed for the creation of simulations based on multi-agent systems. It was implemented on the POP-11 language [1] whose interpreter, called Poplog (see [19] for an interesting history of Poplog and POP-11), supports various programming paradigms and provides many features supporting the development of artificial intelligence applications (e.g., the ease for definition and use of lists and pattern matching). The Poplog interpreter can be downloaded from URL `http://www.cs.bham.ac.uk/research/poplog/freepoplog.html`. The approach they have followed in SIM_AGENT is to provide a tool that is quite general, leaving to the programmer the task of determining the agent architecture. It was, in fact, aimed as a "testbed for cognitively rich agent architectures".

The toolkit was built based on these previous extensions to the POP-11 language:

Objectclass: which supports object oriented programming;
Poprulebase: which provides an inference system (i.e, it supports logic programming);
Rclib: a graphics library for using the X Window System (which is optional).

3.1 Knowledge and Reasoning

The basis for agent reasoning in SIM_AGENT are Poprulebase *rules*. The knowledge of the world within agents is modelled in a *database* as lists of words (in POP-11 syntax, lists are enclosed in square brackets). For example:

```
[colour blue]
[rs brazil state_in]
```

The format and the meaning of the words within lists is defined exclusively by their context, at the user's choice. In our specific case (as seen in Section 4.1), we have chosen to use the first word in a database item to denote a modality (e.g. belief), the second to denote a predicate, and the others are terms standing as parameters to the predicate. Elements of a list can also be a list rather then a single word.

The POP-11 interpreter provides then a *matcher* which unifies a pattern given as parameter with database items (lists of words). In these patterns, the following constructs can be used:

`=` stands for any element from the matching list;
`==` stands for any number of elements;
`?var` states that variable `var` should be unified with one element from the list;
`??var` states that variable `var` could be unified with any number of elements (in this case, the variable is instantiated with a list of those elements).

Rules have the following format:

```
RULE <name> [<CONDITION>] ... ==> [<ACTION>] ...
```

The *action* part of the rule will only execute if the matching of the patterns in the *condition* part of the rule against database items succeeds. As an example, consider the following database and rule:

```
/* Database */
[green watermelon]
[yellow banana]

/* Rule */
RULE example
[LVARS x]                   ;;; states that x will be used
[green ?x]                  ;;; as variable for unification
==>
[SAY The green object is ?x] ;;; prints words and variables
```

3.2 Architecture of a SIM_AGENT Agent

Agents built with SIM_AGENT have the same basic characteristics that are usual in multi-agent systems. It offers a default mechanism for sensing (i.e., perception), which can be redefined for agents with alternative perception approaches. Both perception of and action on the environment are defined as POP-11 procedures.

Reasoning is a distinguishing aspect of agent modelling in SIM_AGENT. It is based on the following structures defined in the Poprulebase library (see Figure 2 for a clear depiction of these structures):

Rule: These are condition/action rules whose conditions are matched (unified) to the agent's database items; rule actions can modify the matching database items or fire agent actions.

Ruleset: A Ruleset is a set of rules which are tested sequentially. There are two operation modes that are relevant to this work: the mode where the ruleset executes all its rules that unify with database items and the mode where a ruleset finishes execution as soon as the first rule is executed (that is, the condition part of the rule unifies with the database).

Rulefamily: A Rulefamily is a set of rulesets in which a single ruleset is active at a given time. At every execution cycle, the interpreter can only run one ruleset (the active one). It can then pass the control over to another ruleset, which will run in the next cycle.

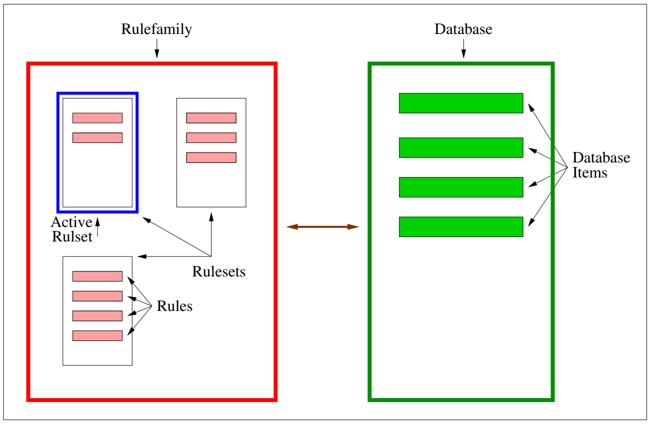


Fig. 2. The Structures Provided by the Poprulebase Library

3.3 Execution Cycle

In the SIM_AGENT toolkit, to put it in a simplified way, an execution cycle consists of the following operations:

1. For each agent in `AgentSet`:
   (a) Run sensor;
   (b) Run the inference system n times (where n is defined for each agent).
2. For each agent in `AgentSet`:
   (a) Execute actions.

As seen above, it is possible to define, for each agent, the number of inference steps over the database it will be allowed at every execution cycle. This allows the execution of various rulesets in a single reasoning cycle of an agent. This mechanism is very important in the interpretation of AgentSpeak(L) programs by SIM_Speak, as seen in the next section.

We here have concentrated on Poprulebase, which underlies SIM_AGENT. The SIM_AGENT toolkit itself provides several other features for the development of agents which we have not described in this section. They were omitted considering that in this paper we focus on agent cognition (recall that AgentSpeak(L) also concentrates on the internal aspects of an agent).

4 SIM_Speak

This section shows how we have made it possible to run AgentSpeak(L) programs on SIM_AGENT. We show how to automatically translate AgentSpeak(L) programs into SIM_AGENT code, and how this code actually makes SIM_AGENT work as an AgentSpeak(L) interpreter (as seen in Section 2.2). This prototype interpreter for AgentSpeak(L) that runs on SIM_AGENT is what we call SIM_Speak.

4.1 Converting an AgentSpeak(L) Agent into a SIM_AGENT Agent

The translation of the initial beliefs specified in a AgentSpeak(L) program is quite straightforward, as it is a simple syntactic change. The translation of plans is more complicated, as explained below. First, we mention the differences between SIM_AGENT and AgentSpeak(L) agents and how to adapt SIM_AGENT to cope with them. We follow the order in which the abstract AgentSpeak(L) interpreter proceeds, and show how the generated SIM_AGENT code copes with each part of the interpretation cycle.

Perception. AgentSpeak(L) and SIM_AGENT have quite different approaches to perception, so we had to adapt SIM_AGENT to cope with AgentSpeak(L)-like perception. In AgentSpeak(L), sensors generate what Rao has called “external events”; they reflect the differences between what an agent believed to be true and its current perceptions. The interpreter actually receives as input (from an external belief revision process based on current perception) a list of events, each one corresponding to one particular change in the agent's belief base. This assumption on the generation of “events” contrasts with some implementations of agents systems; in SIM_AGENT, for example, sensors are procedures which simply add items to the database (with all true statements about the world). To make these two different approaches to perception compatible, we have implemented a ruleset which is in charge of revising beliefs and generating the appropriate events. It starts from previous beliefs about the world, takes new beliefs from perception, and compares them for discrepancies, which are translated into external events. After that, it updates the database with the changed beliefs about the world.

Event Selection. If the agent senses any changes in the world, then there will be `[event ...]` entries in the database. The event selection function ($\mathcal{S}_\mathcal{E}$) was implemented as a ruleset whose rules modify the form of the chosen event so that it fires the rules in another ruleset. The selected event is then changed to `[e ...]` (this marks the event that is currently being dealt with). In this implementation of AgentSpeak(L), we have chosen to treat incoming events as a queue, which should be acceptable for agents in general. Recall that $\mathcal{S}_\mathcal{E}$ is part of the definition of an agent, so that, in principle, agents could have more sophisticated event selection functions, e.g., one that prioritises the handling of certain urgent events. As we mentioned before, AgentSpeak(L) itself does not include the mechanisms to specify agents' selection functions.

Plan Activation. Once a given event is selected, we try to find a plan that deals with that event. In SIM_Speak, each plan is translated into a SIM_AGENT rule, and an intention (i.e., a stack of partially instantiated plans) is represented as a ruleset. The conditions of such plan rule are the triggering event and the context parts (i.e., the whole head) of a plan. The actions that are executed when a plan rule is fired are the following:

– deletion of the triggering event (considering that it will be dealt with by the chosen plan);
– translation of the body of the plan into Poprulebase rules;
– pushing the body of the plan on the top of an existing intention or creation of a new intention.

When a plan is activated, the action part of the rule which is used to represent the plan is in charge of translating the whole body of that plan into other rules which will form a ruleset representing an intention (or will be pushed into a ruleset for an existing intention).

Body of Plans. Each element in the body of a plan is implemented as a rule with an empty condition part, that is, rules that always execute. For each different type of formula in the body of a plan, we have established a different type of rule action, as follows:

Action: The rule calls an external (POP-11) procedure and deletes itself.

Achievement goal: An internal event is generated.

Test goal: The rule unifies the variables for the given intention[1]. These bindings of variables to constants, resulting from pattern matching with the beliefs, which are physically stored in the database, are stored temporarily in the database too. This rule also deletes itself when it finishes.

Belief Addition/Deletion: In this case, all that has to be done is update the database by adding/removing the item representing that belief, and adding the appropriate event item to the database.

[1] Note that unification is also commonplace in SIM_AGENT. This one of the aspects that has greatly facilitated our work of implementing an AgentSpeak(L) interpreter using SIM_AGENT. We mention a few others towards the end of this section.

True: The true intention signals the end of a subplan. It deletes itself and also deletes the rule that generated the triggering event of that subplan. Note that each plan generated by an internal event must end with the true intention. This is assured at the time the rules are built.

The inference system provided by Poprulebase allows the definition of rulesets and rulefamilies, but it does not allow for their dynamic alteration (at least up to version 15.53). We have created some procedures for the dynamic manipulation of those structures, so that we could represent intentions (which are stacks) by means of rulesets. They do all the work of creating new rulesets inside rulefamilies, push subplans, and popping the first formula (i.e., a rule) in the body of plans.

For each current intention of an agent, there exists a record of its name (as a ruleset) in the database. Through these records, the scheduler can access the intentions the agent has at a given time. Whenever an intention is created, we insert at the bottom a "self-destruction" rule. It is responsible for deleting the intention after all its actions have been executed. It also deletes its entry in the records of intention names in the database.

As the unification of the triggering event and the context part of the head of a plan are done together (given that the unification of the head of a plan actually corresponds to the activation of a rule), the order in which two plans for the same triggering event is specified is relevant. The Poprulebase interpreter checks the rules in a ruleset sequentially until it finds a rule for which all conditions unify with database items. If two plans are triggered by the same event, with different contexts, the first unifying context will determine the choice of plan. Referring back to Definition8, this means that we have also used a simple fixed $\mathcal{S}_{\mathcal{O}}$ function (the one that selects one plan from a set of applicable plans), which again could in principle be agent-specific. In fact, this corresponds to the intuition used in logic programming, where the first matching goal clause is chosen.

Finally, we have stored the binding of variables in the database to facilitate the implementation of test goals. We can use the unification provided by SIM_AGENT when we create an intention, because we use the matcher against the database for unifying the triggering event and the context. However, test goals create bindings whilst a plan is being executed, so in this case the bindings have to be annotated somewhere else, otherwise a complex manipulation of rules within a ruleset would have to be implemented.

Format of Database Items. In SIM_AGENT, database items do not have a fixed interpretation, that is, their meaning is completely defined by the programmer (as mentioned before). The system sees each item as an arbitrary object (one usually uses words or lists), which are matched with patterns at the time the inference system is run. Since all our data structures representing the different mental attitudes in the BDI approach are stored in that single database, we defined the following format of items indicating the different sets to which each element belongs:

`[bel ...]` for beliefs;
`[event ...]` for events;
`[e ...]` is a special case denoting the single selected event;
`[int ...]` is used for the names of the rulesets associated with current intentions;
`[var ...]` is used for recording the binding of variables within a given intention;
`[new ...]` for data received by the sensors.

The set of plans is, in fact, a ruleset, as intentions are. Regarding the intentions, we emphasise again that clauses of type `[int ...]` are used only for recording the names of the rulesets storing current intentions.

4.2 A SIM_Speak Interpretation Cycle: Running an AgentSpeak(L) Agent

In SIM_AGENT, by default, only one ruleset can run at each execution cycle (see section 3.3). As our implementation of AgentSpeak(L) requires more than one ruleset, we configure SIM_AGENT so that each agent is allowed five inference steps when it runs. Recall again that within a rulefamily, only a single ruleset can be active at a time, and it can passes the control over to another ruleset at the end of its execution. We next give descriptions for each of the five steps in a SIM_Speak execution cycle. They can be best understood by referring to Figure 3.

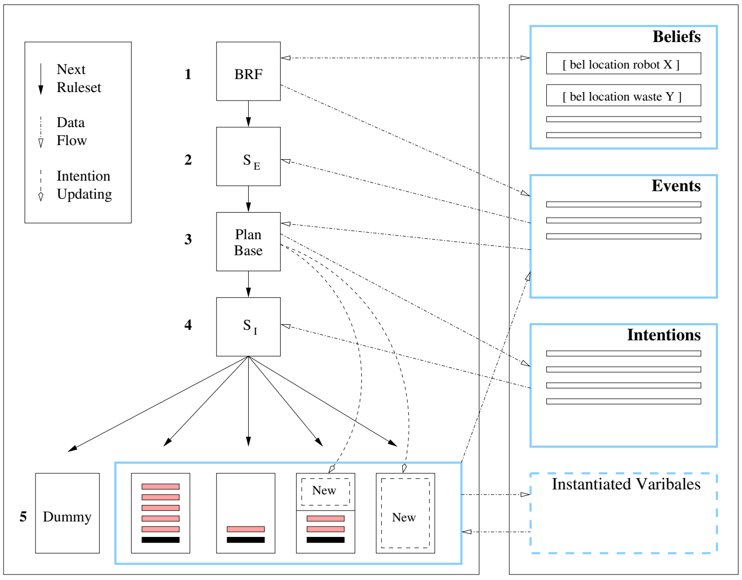


Fig. 3. The Steps in a SIM_Speak Execution Cycle

1. **BRF:** This function reads the database items with new data from the sensors. It revises the beliefs and identifies the events that need be generated (updating the database accordingly).
2. **$\mathcal{S}_\mathcal{E}$:** In this step, one of the events is selected. Recall that, in our implementation, events are selected in a first-in first-out basis.
3. **Plan Library:** Generates an executable body for the plan that was triggered by the selected event, which has to have a valid context too. As mentioned before, the function $\mathcal{S}_\mathcal{O}$ chooses an applicable plan according to their order in the specification of the AgentSpeak(L) source code.

4. **$\mathcal{S}_{\mathcal{I}}$:** This function chooses an intention based on database items with a `[int ...]` format. It then passes the control over to a chosen intention at the end of this inference step. If there is no intention to be executed, the control is passed over to a "dummy" intention (i.e., a ruleset that does nothing), as the fifth allocated reasoning cycle must be run.
   Again we have used a very simple mechanism to stand for all agents' intention selection function. It works as a "round-robin" scheduler, where all separate intentions (i.e., all focuses of attention) are kept in a circular queue. When an intention is selected, it is allowed to execute only the formula in the beginning of the body of the plan that is on the top of that intention (a stack of partially instantiated plans).
5. **Intention Execution:** Now the selected intention or the dummy ruleset is run. Each element in an intention is a rule with no conditions, thus the rule that is the first element in an intention is always executed. Except for achievement goals, all other types of rules delete themselves. If there is a basic action to be executed, which is actually a POP-11 procedure, it is called from the rule.

Note that the ruleset implementing the plan library and those implementing intentions must be configure to stop execution as soon as the first matching rule is fired.

To round off this section, we emphasise that SIM_AGENT has significantly facilitated the work of implementing an AgentSpeak(L) interpreter. Some necessary algorithms (e.g., for unification, checking for logical consequence, and belief revision) were all either available or relatively easy to implement with SIM_AGENT.

5 Discussion

5.1 Current Implementation

As defined by Rao, AgentSpeak(L) was just an abstract language. We have made it practical with the choice of generic selection functions: inserting new events in a queue, considering the specified order of the plans, and scheduling intentions in a round-robin fashion. Neither Rao [15] nor d'Inverno and Luck [5] proposed the means for the specification of such selection functions. It would be very restricting if they were to be implemented by users of AgentSpeak(L) in ordinary programming languages. We also do not have a solution for this yet. In SIM_Speak, more sophisticated selection functions have to be implemented as POP-11 code to be included in the appropriate rulesets.

SIM_Speak is currently implemented as a console based program that receives an AgentSpeak(L) specification and generates a POP-11 source code defining an agent class derived from the basic SIM_AGENT class. In order to have a complete multi-agent system, the user has to define the agent's sensors and effectors for completing an agent's class (that is, the user has to define the perceptions and actions to which agents of that class have access). They are implemented as POP-11 procedures and follow certain rules which can be found in the system's documentation. The SIM_Speak platform is available for downloading at URL `http://www.inf.ufrgs.br/~bordini/SIM_Speak/`.

In SIM_Speak, agent's reasoning is completely specified by AgentSpeak(L) beliefs and plans, exactly the way they were defined by Rao in [15]. SIM_Speak does not require any syntax change, as we provide an automatic translator from AgentSpeak(L) syntax into SIM_AGENT code. The complete implementation of a multi-agent system, however, does require some extra coding in POP-11 using SIM_AGENT, not only for redefining perceptions and actions, but also for starting the execution of the agents (although this is straightforward).

However, there are a few limitations on the SIM_Speak implementation. At this point of its development, triggering events cannot have uninstantiated variables. Although it makes programming slightly more cumbersome, this is not much of a restriction, as we can use the agents own belief base to store the results of a subgoal and use test goals to retrieve them after the subgoal is achieved. In a way, our extension of the language which allows addition and deletion of beliefs also makes up for the limitation related to uninstantiated variables in achievement goals.

As for the assessment of SIM_Speak, we have some successful practical experience with its use. Initially, a variation of the mobile robot toy problem given in Rao's paper [15] was implemented. It consists of a simple multi-agent system, with less than 10 agents, and defines a traffic-world with a waste collecting robot that tries to avoid the cars in a highway, while trying to collect the waste papers that drivers throw on the road. Afterwards, we worked on a large agent application, in order to test the robustness of SIM_Speak—in fact, the robustness of the underlying SIM_AGENT. We have found that we can indeed run large systems of multiple AgentSpeak(L) agents with SIM_Speak. The referred application, used to test SIM_Speak, was a traffic simulation where BDI agents decide which route to take in order to reach a desired location. They make such decisions based on their past experience and their "personalities". This application is presented in detail in [17], including an AgentSpeak(L) specification. We have been able to run a simulation of 2,000 agents, each having approximately 20 plans. This shows that SIM_AGENT is quite robust, and suitable for demanding applications. However, SIM_Speak is arguably not suitable for real-time applications, although its performance has not as yet been assessed.

5.2 Extensions to AgentSpeak(L)

In our recent experience with using AgentSpeak(L) as a specification language for BDI agent applications (see [17]), we have felt the need to include, in the body of plans, addition and deletion of beliefs (which, remember, can be triggering events). This has greatly facilitated the task of creating certain plans. The intuition behind it is that the performing of actions and achievement of subgoals can themselves lead the agent to new beliefs, rather then new beliefs being only possible through perception of the environment as effected by agent actions (and the actions of other agents in the environment, of course). Rao had in fact pointed out to such operations within a plan body, but their definition was not present in his formal semantics. We have other projects for which we have defined a series of other extensions to AgentSpeak(L), e.g., for handling plan failure, using speech-act based communication, and using decision-theoretic scheduling for the automatic generation of efficient, agent-specific intention selection functions.

We are also considering an extension to AgentSpeak(L) that relates to Cohen and Levesque's notion of *persistent goals* [3]. Because of the way AgentSpeak(L) is implemented on SIM_AGENT, we could easily check whether goals that started off the execution of plans have been achieved before the completion of their execution. However, it is not so straightforward to deal with dropping plan executions because goals are no longer achievable.

5.3 Related Work

There is presently a variety of approaches to agent programming. We do not aim at a comprehensive account of them here; we shall concentrate on a few BDI languages that are well-known and the ones that are discussed in other papers in this volume.

One of the best-known BDI languages in the multi-agent systems community is 3APL [8]. It is quite similar to AgentSpeak(L) and in fact has a few more sophisticated language constructs (e.g., for handling plan failure). Although these were not present in the semantics of the language [15,5], Rao pointed out to the syntactical constructs that can be used to expand AgentSpeak(L) in respect to some of the 3APL features (we have mentioned above a few of the extension we have already conceived). However, as AgentSpeak(L) in its definition, 3APL is an abstract programming language, with no available interpreter, it appears. Other BDI languages do have implemented interpreters, but appear to have a less strong theoretical foundation (e.g., JAM [9]).

In AgentSpeak(L), an agent is specified simply by a set of initial beliefs and a set of plans (although there is the issue of specific selection functions that may be required), and it has a neat notation for BDI abstractions. It is simpler than $\mathcal{MINERVA}$ [12], a logic programming approach that uses an update command language (for dynamic logic programming) to define agent structures. Although abstract in its original definition, AgentSpeak(L) is intended as a practical programming language. In [10], Kinny presents the Ψ Calculus, an algebraic language for agents based on Milner's π-Calculus [13] and accounting for BDI abstractions. Although a clearly important theoretical work, a practical programming language is yet to be defined based on his Ψ Calculus.

6 Conclusion

We hope to have increased the interest in AgentSpeak(L) by actually providing the means for running AgentSpeak(L) programs. We have also shown the value of AgentSpeak(L) as a specification language for cognitive (BDI) multi-agent systems in [17]. We expect that the multi-agent systems research community will reconsider the use AgentSpeak(L), and give it the merits it deserves. As a purely abstract programming language, it has been rightfully dismissed by all practitioners in the area. This needs no longer be the case, as it is now fully operational, although the definition of selection functions (when required), as well as the coding of the environment shared by the agents, still requires some knowledge of the SIM_AGENT toolkit and the underlying POP-11 programming language.

We have implemented an automatic translator from AgentSpeak(L) syntax into code that is ready for execution in SIM_AGENT. As seen in Section 4, SIM_AGENT greatly facilitated the development of mechanism that allows us to run AgentSpeak(L) programs. SIM_AGENT is in fact a quite useful toolkit, and we aim at using SIM_Speak as the basis of a system for the creation of cognitive multi-agent social simulations. That system is intended for the use of social scientists, and therefore should not involve actual programming of the simulations but allow their creation very easily.

Although SIM_Speak only runs on SIM_AGENT, it should turn out to be a suitable platform for the development of many applications, maybe except for those which have to deal with real-time constraints. For applications where performance is indeed essential, we are developing a purpose-build interpreter for AgentSpeak(L). This interpreter follows closely the formal definition of an AgentSpeak(L) interpreter given by Rao in [15]. This way, we should keep the one-to-one correspondence with the proof system defined by Rao, which is the solution he proposed for the undesirable divide between BDI theory and practice. One of the difficult tasks in the development of such interpreter will be to deal with aspects of modelling environments and the basic actions that agents can perform, as this is not defined formally in AgentSpeak(L); the language concerns exclusively the internal aspects (i.e., the mental attitudes) of agents.

References

1. James A. D. W. Anderson, editor. *POP-11 Comes of Age: The Advancement of an AI Programming Language*. Ellis Horwood, Chichester, U.K., 1989.
2. Rafael H. Bordini, John A. Campbell, and Renata Vieira. Extending ascribed intensional ontologies with taxonomical relations in anthropological descriptions of multi-agent systems. *Journal of Artificial Societies and Social Simulation*, 1(4), October 1998. <http://www.soc.surrey.ac.uk/JASSS/1/4/3.html>.
3. Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.
4. Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge. A formal specification of dMARS. In Munindar P. Singh, Anand S. Rao, and Michael Wooldridge, editors, *Intelligent Agents IV—Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97), Providence, RI, 24–26 July, 1997*, number 1365 in Lecture Notes in Artificial Intelligence, pages 155–176. Springer-Verlag, Berlin, 1998.
5. Mark d'Inverno and Michael Luck. Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation*, 8(3):1–27, 1998.
6. Gregory Duck and Leon Sterling. Prototyping from a specification—an agent-based case study. Unpublished paper, available at URL `http://www.cs.mu.oz.au/~gjd/research/`.
7. Michael P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87), 13–17 July, 1987, Seattle, WA*, pages 677–682, Manlo Park, CA, 1987. AAAI Press / MIT Press.
8. Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Control structures of rule-based agent languages. In Jörg P. Müller, Munindar P. Singh, and Anand S. Rao, editors, *Intelligent Agents V—Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), held as part of the Agents' World, Paris, 4–7 July, 1998*, number 1555 in Lecture Notes in Artificial Intelligence, pages 381–396, Heidelberg, 1999. Springer-Verlag.
9. Marcus J. Huber. JAM: A BDI-theoretic mobile agent architecture. In *Proceedings of the Third International Conference on Autonomous Agents (Agents-99), 1–5 May, Seattle, WA*, pages 236–243. ACM Press, 1999.
10. David Kinny. The Ψ calculus: an algebraic agent language. (In this volume).
11. David Kinny. The distributed multi-agent reasoning system architecture and language specification. Technical report, Australian Artificial Intelligence Institute, Melbourne, Australia, 1993.

12. João Alexandre Leite, José Júlio Alferes, and Luís Moniz Pereira. $\mathcal{MINERVA}$—a dynamic logic programming agent architecture. (In this volume).
13. Robin Milner, Joachim Parrow, and David Walker. A calculus for mobile processes (parts I and II). *Information and Computation*, 100(1):1–40 and 41–77, September 1992.
14. Gordon D. Plotkin. A structural approach to operational semantics. Technical report, Computer Science Department, Aarhus University, Aarhus, 1981.
15. Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Walter Van de Velde and John Perram, editors, *Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), 22–25 January, Eindhoven, The Netherlands*, number 1038 in Lecture Notes in Artificial Intelligence, pages 42–55, London, 1996. Springer-Verlag.
16. Anand S. Rao. Decision procedures for propositional linear-time belief-desire-intention logics. In Michael Wooldridge, Jörg P. Müller, and Milind Tambe, editors, *Intelligent Agents II—Proceedings of the Second International Workshop on Agent Theories, Architectures, and Languages (ATAL'95), held as part of IJCAI'95, Montréal, Canada, August 1995*, number 1037 in Lecture Notes in Artificial Intelligence, pages 33–48, Berlin, 1996. Springer-Verlag.
17. Rosaldo J. F. Rossetti, Rafael H. Bordini, Ana L.C. Bazzan, Sergio Bampi, Ronghui Liu, and Dirck Van Vliet. Using BDI agents to improve driver modelling in a commuter scenario. *Transportation Research Part C: Emerging Technologies*, 2002. To appear.
18. Munindar P. Singh, Anand S. Rao, and Michael P. Georgeff. Formal methods in DAI: Logic-based representation and reasoning. In Gerhard Weiß, editor, *Multiagent Systems—A Modern Approach to Distributed Artificial Intelligence*, chapter 8, pages 331–376. MIT Press, Cambridge, MA, 1999.
19. Aaron Sloman. The evolution of Popolog and POP-11 at Sussex University. In James A. D. W. Anderson, editor, *POP-11 Comes of Age: The Advancement of an AI Programming Language*, pages 30–54. Ellis Horwood, Chichester, U.K., 1989.
20. Aaron Sloman and Brian Logan. Building cognitively rich agents using the SIM_AGENT toolkit. *Communications of the Association of Computing Machinery*, 43(2):71–77, March 1999.
21. Aaron Sloman and Riccardo Poli. SIM_AGENT: A toolkit for exploring agent designs. In Michael Wooldridge, Jörg P. Müller, and Milind Tambe, editors, *Intelligent Agents II—Proceedings of the Second International Workshop on Agent Theories, Architectures, and Languages (ATAL'95), held as part of IJCAI'95, Montréal, Canada, August 1995*, number 1037 in Lecture Notes in Artificial Intelligence, pages 392–407, Berlin, 1996. Springer-Verlag.
22. J. Michael Spivey. *The Z Notation: A Reference Manual.* Prentice Hall, Hemel Hempstead, second edition, 1992.
23. Michael Wooldridge. Intelligent agents. In Gerhard Weiß, editor, *Multiagent Systems—A Modern Approach to Distributed Artificial Intelligence*, chapter 1, pages 27–77. MIT Press, Cambridge, MA, 1999.
24. Michael Wooldridge. Computationally grounded theories of agency. In Edmund Durfee, editor, *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000), 10–12 July, Boston*, pages 13–20, Los Alamitos, CA, 2000. IEEE Computer Society. Paper for an Invited Talk.

Ontological Overhearing

Marco Aiello[1], Paolo Busetta[2], Antonia Donà[2], and Luciano Serafini[2]

[1] ILLC and ISIS, University of Amsterdam
Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands
and
DISA-University of Trento, Via Inama 5, 38100 Trento, Italy
aiellom@ieee.org

[2] ITC-IRST, Via Sommarive 18, 38050 Povo, Trento, Italy
{busetta, antodona, serafini}@itc.it

Abstract. The collaboration between two intelligent agents can be greatly enhanced if a third agent, who has some understanding of the communication between the first two, intervenes giving appropriate information or acting helpfully without having been explicitly involved. The behavior of this third agent, quite common in human interaction, is called *overhearing*. We present an agent architecture modeling this behavior. In particular, we focus on overhearing based on ontological reasoning; that is, the overhearer semantically selects pieces of communication according to his own knowledge (ontologically organized) and goals. In our architecture, overhearing is performed by a team of agents playing two different roles: the first role (overhearer) classifies the overheard communication according to a formal ontology; the second role (suggester) makes appropriate suggestions at the appropriate time point. We present a formal language for the interaction between agents in the overhearing team. A prototype of the architecture, implemented using JACK Intelligent Agents, is briefly described and preliminary experimental results are discussed.

1 Introduction

Humans work well in teams. In a collaborative environment, whenever people are faced with tasks that they cannot manage, or know that can be managed better by others, they seek assistance. This observation is not new, and has inspired much research in cooperative agents, for example [15,25,22].

We choose to analyze teamwork through a slight shift in perspective. While association between agents can readily be achieved by requesting help when needed, equal or even improved results can be achieved when associates observe the need for help, and initiate actions or offer suggestions with the aim of improving the plight of their colleague. In fact, these associates may communicate not only when a colleague needs assistance, but also when they feel they can help improve the productivity of their team, or when they believe to achieve a personal gain from the suggestion.

As part of our work on frameworks for collaboration, we have introduced an abstract architecture based on a principle called *overhearing* [12]. The intuition behind overhearing comes from the modeling of such human interaction as aforementioned, in a collaborative observable environment. The overhearing architecture describes how non-planned collaboration in a community of artificial agents can be achieved by means of *unobtrusive observations* and *unsolicited suggestions*.

This paper focuses on a single aspect of the architecture: the observation of the conversations between two or more agents. Our goal is to provide a framework for querying a communication channel on the development of a conversation; queries are typically submitted by agents willing to provide unsolicited assistance. To this end, we defined a formal language, designed the software components required for its interpretation, and performed some experiments on a real-life, multi-agent, observable channel (a web-based newsgroup). Our formal language includes both temporal and ontological components, and allows the formulation of complex queries on contents, performatives, and order of the messages being exchanged.

The paper is organized as follows. Next section gives an overview of the overhearing architecture. In Section 3 we provide the definition of the language for querying the communication channel, and show how formulas of the language are interpreted. In Section 4, we present the use of the formal language within the overhearing architecture. An actual implementation and some experimental results, showing the effectiveness of the proposed framework, are presented in Sections 5 and 6, respectively. Section 7 discusses some related work. We conclude by identifying potential future research (Section 8).

2 The Overhearing Architecture: An Overview

The overhearing abstract architecture is summarized in Figure 1. Service agent A and service agent B are communicating over a channel and observed by the overhearing agent (*overhearer* from here on). A suggester agent (*suggester*) subscribes with the overhearer to be notified if a certain type of event occurred on the channel. Finally, the suggester is able to issue *suggestions* to any of the service agents; a suggestion is a special message carrying information or commands. In general, a running system contains many couples of agents covering the role of services, more than one agent acting as suggester, and at least one overhearer; an agent may cover more than one role simultaneously (such as service and suggester).

Overhearing differs from blackboard-based architectures (see for instance, the Open Agent Architecture [13]) because it is not concerned with connecting services, nor controlling the flow of messages. Our aim is not to provide yet another communication facility, rather to support a flexible development methodology for complex adaptive systems. In the initial phases of development, only those agents (services in the above terminology) required to achieve the basic functionality should be built and their interactions fully specified; for instance, these could be negotiating agents working on some allocation problem (see [16,17,26,6] in this volume). The behavior of these services, however, should be modifiable by external observers via suggestions. While functionality of the basic services required by an application are assumed to be immutable, suggesters may be dynamically added and removed, without hampering the ability of the system to reach its main objectives.

This approach has various advantages. Firstly, it is possible to enhance functionality of a running system. As an example, state-of-the-art machine-learning or tunable components can be plugged into the system, as and when they become available, without the

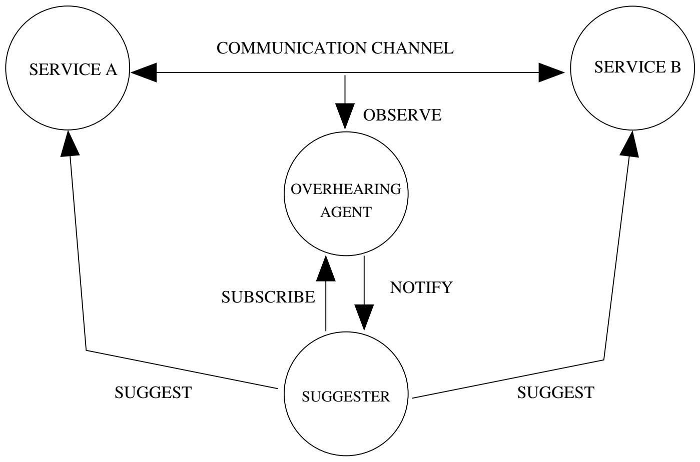


Fig. 1. Overhearing architecture.

need to bring down, rebuild, and then restart the system. Secondly, the output of a system can be enhanced either by suggesting additional related information, or requesting the deletion of outdated or unrelated results.

This flexibility comes at a cost. Indeed, a full implementation of the overhearing architecture requires the suggester to be able to perform agent state recognition, based on a model of the service agents and the messages they exchange. Communication—or at least some selected conversations—needs to be observable, e.g., by using a broadcast service, and this in turn may introduce issues with performance, timing and security. Last but not least, services need to be engineered to handle suggestions and to change their behavior accordingly.

Overhearing has been applied to various case studies, from the simple demonstrator presented in [12], to a more interesting agent-based Web server where suggesters can send additional data to assistant agents build dynamic HTML pages from relational databases. Information that does not easily fit into a relational schema (e.g., unstructured text, targeted advertising based on dynamically built user profiles) is collected by suggesters, and sent to the assistants whenever appropriate. Overhearing can be seen as one of the possible ways to implement the paradigm of "implicit culture" for agent collaboration [8]. Other potential applications may include the analysis of negotiation protocols (see for instance [17] in this volume), possibly to detect antisocial behaviour [11].

In what follows, we concentrate on the interaction between suggester and overhearer. As mentioned above, the objective is for suggester to be notified of all and only those messages that are relevant to him.

3 Overhearer—Suggester Interaction Language

In the architecture presented above, an overhearer has two main goals: monitoring a communication channel, and responding to the queries of suggester agents about the conversations taking place.

Depending on the application, messages exchanged by services may vary from fully structured (e.g., most client/server interactions, auction, etc.), to semi structured (e.g., XQL queries and XML pages), to unstructured ones (e.g., the body of email or newsgroup messages). Often, content is in natural language; sometimes it may even be of a multimedia nature, such as images and sound tracks. In many situations, it is not feasible for an overhearer to keep track of the entire content of all messages. In case of intensive communication, indeed, this simply requires too much memory space. An overhearer keeps track of the conversations by logging a suitable amount of data for each message traveling on the channel, that is a "summary" representing an *interpretation* of the message with respect to a formal domain ontology.

Suggesters that want to be notified of the messages regarding specific topics can subscribe to the overhearer. In its subscription, a suggester specifies a matching criterion (a pattern), which the overhearer has to apply to select the messages to forward.

To provide a reliable and well founded tool for expressing such a selection criterion, we developed a multi-modal language inspired by description logics and modal temporal languages. We called the language $\mathcal{PT}_{\mathcal{ALC}}$ since it is a modal $\mathcal{T}$emporal logic over a simple description language of the $\mathcal{ALC}$ family enriched with $\mathcal{P}$erformatives. Figure 2 highlights the use of $\mathcal{PT}_{\mathcal{ALC}}$ for subscriptions, and the main facilities used by the overhearer: a list of subscriptions, an internal clock to time the channel, and a log file for the messages. As discussed in the sequel, the latter contains message interpretations in $\mathcal{P}_{\mathcal{ALC}}$, which is a subset of $\mathcal{PT}_{\mathcal{ALC}}$ without temporal operators.

Suppose a suggester's goal is to bring a certain castle, the Buonconsiglio castle in Trento, to the attention of potential tourists of the Trentino region. This can be achieved in various ways. For instance, the suggester could ask to be informed of all the messages passing on a public channel between an agent browsing the web and an agent serving web pages; or, he could ask to be informed of all the messages containing a specific set of words (e.g., {Castle, Trento, Tourism}). Alternatively, he could ask to be informed whenever a message containing concepts related to castles and tourism has been uttered. The latter appears to be the most appropriate approach, therefore we require the overhearer—suggester interaction language to express structured concepts. In addition, a temporal dimension is necessary to express properties of the temporal order of messages.

The suggester can ask if an agent has uttered a request regarding castles located in Trentino, or regarding a particular castle located in Trentino, in the following way:

$$\Diamond_p(\textsc{Ask}(*, *, \mathit{castle} \sqcap \forall \mathit{is\_located}.\mathit{Trentino})) \ . \tag{1}$$

The suggester is expressing information of a very different nature with the above formula. Let us analyze it bottom-up:

– **Conceptual:** $\mathcal{ALC}$**.** $\mathit{castle} \sqcap \forall \mathit{is\_located}.\mathit{Trentino}$ The suggester is expressing the concept of things that are both castles and that for all roles "is_ located" have a filler of type Trentino. We denote this formula by φ.

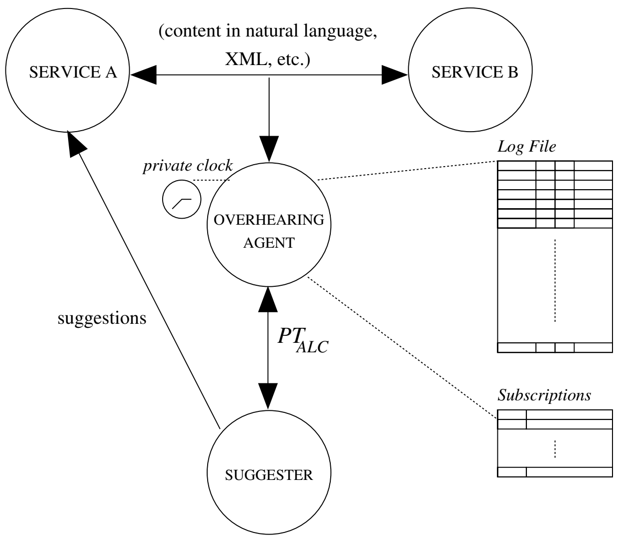


Fig. 2. Overhearing architecture: the interaction language and facilities of the overhearer.

- **Performative:** $\mathcal{P}_{\mathcal{ALC}}$. $\textsc{Ask}(*, *, \varphi)$ The suggester is interested in performance of the type $\textsc{Ask}$ from any agent to any other agent. The $*$ stands for a wild-card, in alternative he could have explicitly referred to the name of agents involved in the channel. We denote this formula by ψ.
- **Temporal:** $\mathcal{PT}_{\mathcal{ALC}}$. $\Diamond_p \psi$ The suggester is interested of performances ψ occurred at some point in the past, expressed by the temporal operator $\Diamond_p$.

In the next three Subsections, we give the precise syntax and semantics of all the components of the language $\mathcal{PT}_{\mathcal{ALC}}$.

3.1 The Conceptual Language

Description logics are a family of formalisms spun off from research in semantic networks (see for instance [28,3]). One of the main advantages of these formalisms over previous approaches is the availability of a precisely defined semantics which ensures correctness of reasoning tasks such as subsumption checking.

The alphabet is composed of three main sorts: one for atomic concept names, one for role names and one for object names. For instance, $castle$ is an atomic concept name: that of all entities being castles. $is\_located$ is a role name: that connecting concepts to other concepts in which they are located. $Buonconsiglio$ is the name of an object: in this case, a physical castle located in Trento.

From the alphabet symbols it is possible to define the set of *concepts* as follows. Notationally, A represents an atomic concept name, C and D denote any concept, and R denotes a role name.

$$C, D ::= A \mid C \sqcup D \mid C \sqcap D \mid \forall R.C \mid \exists R.C \ . \tag{2}$$

Relations among concepts, and relations among objects and concepts can be expressed by means of two types of *formulas*:

– $o : C$ is an instance selection,
– $C \sqsubseteq D$ is a subsumption statement.

Formulas of $\mathcal{ALC}$ are denoted by small Greek letters, ϕ, ψ, ... Using the syntax above, we can specify that $Buonconsiglio$ is an object (*instance*) of the concept of things which are $castle$s and are located somewhere in the province of $Trentino$:

$$Buonconsiglio : castle \sqcap \forall is\_located.Trentino\ . \tag{3}$$

We can also state that the concept of castle is less general than the concept of large building, by the formula

$$castle \sqsubseteq building \sqcap large\ . \tag{4}$$

The semantics is given in terms of interpretations I. An interpretation is a pair composed of a *domain* Δ^I, and an *interpretation function* $\cdot^I$ that assigns a subset of Δ^I to each concept name, a subset of $\Delta^I \times \Delta^I$ to every role name, and an element of Δ^I to each object name. The interpretation I for all the concepts is defined as follows:

$$(C \sqcup D)^I = C^I \cup D^I \tag{5}$$
$$(C \sqcap D)^I = C^I \cap D^I \tag{6}$$
$$(\forall R.C)^I = \{p \in \Delta^I \mid \text{for all } q \in \Delta^I,\ \langle p, q\rangle \in R^I \text{ implies } q \in C^I\} \tag{7}$$
$$(\exists R.C)^I = \{p \in \Delta^I \mid \text{there is a } q \in \Delta^I, \text{ such that } \langle p, q\rangle \in R^I \text{ and } q \in C^I\}\ . \tag{8}$$

Formulas can be verified or falsified by an interpretation I according to the following rules:

$$I \models o : C \text{ if and only if } o^I \in C^I \tag{9}$$
$$I \models C \sqsubseteq D \text{ if and only if } C^I \subseteq D^I\ . \tag{10}$$

A *terminological box*, (*TBox*) is a set of subsumption statements; an *assertional box* (*ABox*) is a set of instance selections. A *TBox* together with an *ABox* form a *knowledge base*. We use Σ to denote a knowledge base. An interpretation is a *model* of a knowledge base Σ if every sentence of Σ is satisfied in the interpretation.

3.2 The Performative Language

The performative level of the language allows us to express intentions in connection with concepts. In general, such intentions can be of very different sorts: asking a question, replying to a comment, requesting an action, etc. In general, we identify a set of performatives Per, which represents all performances that can be recognized by the overhearer from the messages passing on the channel. For the sake of simplicity, we concentrate on the two performatives ASK and TELL, representing the performance of a query and the performance of an assertion, respectively. The set of *message patterns* of $\mathcal{P}_{\mathcal{ALC}}$ is any object of the form:

$$\textsc{Per}(i, j, \phi) \tag{11}$$

where $\textsc{Per}$ is either $\textsc{Ask}$ or $\textsc{Tell}$, i, j are either names of agents or the wild-card $*$, and ϕ is either an object name or a concept, or an instance selection, or a subsumption statement. Message patterns are denoted by small Greek letters μ, ν. We also denote the set of agent names with Ag. Message patterns without wild-cards are called *messages*.

Intuitively, the message pattern $\textsc{Ask}(i, *, C)$ represents the set of messages which are queries posted by agent i to any other agent, concerning the concept C. The formula $\textsc{Ask}(i, *, o : C)$ represents the set of messages which are queries posted by agent i to any other agent, concerning the object o as an instance of a concept C. To formalize this intuition we need to extend the semantics of $\mathcal{ALC}$. Formally: an interpretation I of $\mathcal{ALC}$ can be extended to an interpretation for $\mathcal{P}_{\mathcal{ALC}}$ in the following way:

$$i^I = \{i\} \text{ if } i \text{ is not a wild-card; } Ag \text{ otherwise} \tag{12}$$
$$(\textsc{Per}(i, j, o))^I = \{\langle \textsc{Per}, x, y, o^I \rangle \mid x \in i^I \text{ and } y \in j^I\} \tag{13}$$
$$(\textsc{Per}(i, j, C))^I = \{\langle \textsc{Per}, x, y, D^I \rangle \mid x \in i^I, y \in j^I \text{ and } I \models D \sqsubseteq C\} \tag{14}$$
$$(\textsc{Per}(i, j, o : C))^I = \{\langle \textsc{Per}, x, y, o^I, D^I \rangle \mid x \in i^I, y \in j^I \text{ and } I \models D \sqsubseteq C\} \tag{15}$$
$$(\textsc{Per}(i, j, C \sqsubseteq D))^I = \{\langle \textsc{Per}, x, y, C^I, E^I \rangle \mid x \in i^I, y \in j^I \text{ and } I \models E \sqsubseteq D\} \tag{16}$$

Subsumption between message patterns is defined in terms of containment of their interpretations. Formally:

$$I \models \mu \sqsubseteq \nu \text{ if and only if } \mu^I \subseteq \nu^I \ . \tag{17}$$

Intuitively $I \models \mu \sqsubseteq \nu$ means that, according to the interpretation I, any message that matches pattern μ also match pattern ν; put differently, pattern μ is more specific (less general) than pattern ν.

3.3 The Full Temporal Language $\mathcal{PT}_{\mathcal{ALC}}$

The last channel phenomenon we want to model is time. We chose the same formalism as since and until logics (see for instance [29]), which gives us the ability to refer to the past, the future and to the next temporal interval (the next utterance), but also to place conditions on future or past events.

The syntax of the temporal language $\mathcal{PT}_{\mathcal{ALC}}$ is as for the previous languages, with the addition of the following operators over formulas of $\mathcal{P}_{\mathcal{ALC}}$: X_f, $\Diamond_f$, $\Box_f$, X_p, $\Diamond_p$, $\Box_p$ as temporal monadic operators, $\mathcal{S}, \mathcal{U}$ (since and until) as temporal dyadic operators and the classical connectives ($\wedge$, $\vee$, and $\neg$). Operators labelled with a p [f] refer to the past [future]. Rather than over propositional letters, all these operators work over message patterns. Therefore, the atomic formulas of $\mathcal{PT}_{\mathcal{ALC}}$ are message patterns. Intuitively the atomic formula composed of the message pattern μ means that a message that matches μ is now passing on the channel. The formula $\mu \wedge \nu$ means that a message that matches both μ and ν is now passing on the channel. The formula $X_p\mu$ means that a message that matches the pattern μ passed on the channel at the previous time stamp. Formulas in $\mathcal{PT}_{\mathcal{ALC}}$ are denoted by Greek letters α, β, etc. Notice that any message pattern μ is a formula (actually an atomic formula) of $\mathcal{P}_{\mathcal{ALC}}$.

Since the objects of our temporal logic are performatives over message patterns, rather than propositional letters that can be true or false at a given time point as in usual logics, we need to specify the semantics for $\mathcal{PT}_{\mathcal{ALC}}$ in terms of Kripke structures on interpretations of message patterns. This semantics is an extension of the Kripke-like ones for since and until logics in the case of linear discrete time, bounded in the past.

A model M or $\mathcal{PT}_{\mathcal{ALC}}$ on the knowledge base Σ is an infinite sequence $M = M(1), M(2), \ldots$ indexed by natural numbers, where for each natural number s, $M(s)$ is a pair $\langle \mu(s), I(s) \rangle$, composed of a message $\mu(s)$ and an interpretation $I(s)$ of $\mathcal{ALC}$ that is a model of Σ. Satisfiability for $\mathcal{PT}_{\mathcal{ALC}}$ is defined as follows:

$$\begin{array}{ll}
M, s \models \nu & \text{iff } I(s) \models \mu(s) \sqsubseteq \nu \\
M, s \models \neg\alpha & \text{iff } M, s \not\models \alpha \\
M, s \models \alpha \vee \beta & \text{iff } M, s \models \alpha \text{ or } M, s \models \beta \\
M, s \models \alpha \wedge \beta & \text{iff } M, s \models \alpha \text{ and } M, s \models \beta \\
M, s \models X_f\alpha & \text{iff } M, s+1 \models \alpha \\
M, s \models X_p\alpha & \text{iff } s > 0 \text{ and }, M, s-1 \models \alpha \\
M, s \models \Diamond_f\alpha & \text{iff for some } t > s, M, t \models \alpha \\
M, s \models \Diamond_p\alpha & \text{iff for some } t < s, M, t \models \alpha \\
M, s \models \Box_f\alpha & \text{iff for all } t > s, M, t \models \alpha \\
M, s \models \Box_p\alpha & \text{iff for all } t < s, M, t \models \alpha \\
M, s \models \alpha\mathcal{S}\beta & \text{iff for some } t < s,\ M, t \models \beta, \text{ and for all } t \leq w < s,\ M, w \models \alpha \\
M, s \models \alpha\mathcal{U}\beta & \text{iff for some } t > s,\ M, t \models \beta, \text{ and for all } s \leq w < t,\ M, w \models \alpha
\end{array}$$

Intuitively, the truth condition $M, s \models \nu$ represents the fact that the message passed at time s (i.e., $\mu(s)$) matches the message pattern ν, according to the current interpretation of the conceptual language given by the overhearer (i.e., $I(s) \models \mu(s) \sqsubseteq \nu$). Connectives are treated classically; truth conditions for temporal operators are the usual conditions for linear past and future temporal logic bounded in the past.

4 Answering Queries from the Suggester

We now focus on the use of $\mathcal{PT}_{\mathcal{ALC}}$ by overhearer and suggester. When the suggester needs to be informed of something regarding the communication among service agents, he formulates the knowledge he wants to gain in terms of a $\mathcal{PT}_{\mathcal{ALC}}$ message pattern. Suppose that the suggester wants to be notified when an agent i asked about castles in Trentino, and nobody answered mentioning the existence of the Buonconsiglio castle. The suggester would then *subscribe* to the overhearer with the following pattern:

$$(\neg\textsc{Tell}(*, i, \mathit{Buonconsiglio}))\mathcal{S}(\textsc{Ask}(i, *, \mathit{castle})) . \quad (18)$$

The intended meaning of this subscription is to *notify the subscribing agent at every time instant for which the subscribed formula is true.* If, for instance, agent i at time t performed $\textsc{Ask}(i, j, \mathit{castle} \sqcap \forall \mathit{is\_located.Trentino})$ and at time $t+1$ no agent performed a $\textsc{Tell}$ to i with $\mathit{Buonconsiglio}$ as content, then at time $t+1$ formula (18) becomes true and the overhearer will notify the suggester.[1] One may imagine that the suggester would then directly send a message to the agent i at time $t + 2$, informing of the existence of the Buonconsiglio castle in Trentino. The informative action of the suggester, if done over the same communication channel observed by the overhearer, would also make formula (18) false at all times greater than $t + 2$, so he will not be notified again. The question is now, how does the overhearer know if and when a given $\mathcal{PT}_{\mathcal{ALC}}$ formula is true?

As shown in Figure 2, the overhearer updates two main data structures: a log file, storing information about the communication on the channel, and a file of the active subscriptions from the various agents.

In order to interpret the messages he logs, the overhearer needs an *ontology*. In the following, we assume this ontology to be a knowledge base Σ as described in Section 3. The log file has a structure similar to the following one:

Key	Time	Sender	Receiver	Performative	Content in φ
1	12:38:59 PM	i	j	Ask	$castle$
2	12:39:07 PM	j	i	Tell	$Buonconsiglio : castle$
3	12:39:08 PM	k	j	Ask	$wheather \sqcap forecast$
.	...	.	.	...	...

and it is updated every time a performance occurred on the channel. The active subscription file is simply a list of couples with a suggester's name and a $\mathcal{PT}_{\mathcal{ALC}}$ subscribed formula, updated every time a suggester subscribes or unsubscribes.[2]

The log file closely resembles a model for $\mathcal{PT}_{\mathcal{ALC}}$. Indeed, we have a family of ψ messages indexed by natural numbers bound both in the past and in the future. The overhearer uses this model to check whether any subscribed formula is true at the current time point. The procedure to follow when a new message has been observed is straightforward:

1. interpret the message using Σ, and update the log file;
2. for all subscribed formulas:
   (a) check if it is true in the current time step
   (b) if it is, notify the corresponding suggester

Various optimizations to the algorithm are immediate. If a formula is never going to be true in the future (see the example at the beginning of this Section), never check it again. If a formula refers to the past (e.g., $\Diamond_p$) keep a trace of its relevant truth values in the past time instants, instead of recalculating the truth at each time step. The overhearer could try to check the truth of a formula at each time step incrementally, just by checking if a new message has modified its truth value.

[1] Note the subsumption of the concept of the castles located in Trentino by the more general concept of a castle, i.e., $\Sigma \models castle \sqcap \forall is\_located.Trentino \sqsubseteq castle$.

[2] In the presented setting, we assume the suggesters do not use formulas with future operators, because the overhearer can not see into the future.

5 From Theory to Practice

We ran preliminary experiments to verify that $\mathcal{PT}_{\mathcal{ALC}}$ is an adequate interaction language between overhearer and suggester. We prototyped the overhearer—suggester interaction using JACK Intelligent Agents [1], a Java-based BDI platform. The overhearer was equipped with a parser, developed with JavaCC [24], for a subset of the performative language $\mathcal{P}_{\mathcal{ALC}}$, chosen as the core of the complete $\mathcal{PT}_{\mathcal{ALC}}$. The overhearer had an ontology, in the form of a set of beliefs, implementing a knowledge base as described in Section 3. Finally, the overhearer was provided with some weak natural language processing (NLP) capabilities (see for instance [5]); most notably, a list of stop-words for Italian.

In order to implement the matching conditions expressed in Equations 12–16 we developed algorithms for instance checking, for checking the identity of instances, and for subsumption checking. The first two are rather trivial; the interesting portion is the subsumption checking. Traditionally, there are two ways to tackle this problem: in the syntactic approach, tableau methods are used (see for instance [21]); in the semantic approach, it is necessary to build a so called “description graph” representing the model of the formula with respect to the knowledge base. We chose to follow the latter, extending the algorithm of Borgida and Patel-Schneider [10]. The extension was necessary because we allow for concept disjunctions $\sqcup$. We embedded also instances (not dealt with in [10]) in the graphs. Unlike the work in [10], we do not allow cyclic axiom definitions. The modifications we made to the algorithm were possible since we deal with small ontologies and small formulas. In fact, the description graphs associated with $\mathcal{P}_{\mathcal{ALC}}$ specifications grow exponentially in the size of the formula and also grow with the size of the ontology.

We tested the prototype on a simple case: Italian newsgroups. A newsgroup can be seen as a communication channel between intelligent agents exchanging messages in natural language. We used the subjects of the messages as the content of the communications, interpreted by the overhearer according to its own ontology and transformed automatically in formulas of $\mathcal{P}_{\mathcal{ALC}}$. In such a setting, the temporal component plays a minor role with respect to the performative-conceptual one. The reason is that the suggester is interested in simple temporal information, that is, whether a message has been uttered in the past or not. Therefore, the experiments which follow focus the attention on the performative-conceptual portion $\mathcal{P}_{\mathcal{ALC}}$. The goal is that of assessing that indeed the language has the appropriate expressive power to enable the suggester to overhear all the messages, and only those, of “conceptual interest” to him.

6 Experimental Results

We launched the prototype on two newsgroups on an Italian web site, called Global News (`http://www.globalnews.it`). The topic of the first newsgroup is meteorology, while that of the second is traveling. We created two different ontologies: one about meteorological phenomena, consisting of 68 concepts (mainly divided in 4 categories: meteorological phenomena, time periods, political geography and orography) and 230 instances; a second one about traveling, consisting of 190 concepts and 519 instances.

id.	Query	Translation
Q1	$\textsc{Ask}(*, *, \top)$	everything
Q2	$\textsc{Ask}(*, *, neve)$	snow
Q3	$\textsc{Ask}(*, *, vento \sqcup (mare \sqcup lago))$	wind or sea or lake
Q4	$\textsc{Ask}(*, *, precipitazioni \sqcap grafici)$	graphs of precipitation
Q5	$\textsc{Ask}(*, *, \top)$	everything
Q6	$\textsc{Ask}(*, *, \top)$	everything
Q7	$\textsc{Ask}(*, *, crociera \sqcup (Caraibi \sqcup Europa)$	cruising
Q8	$\textsc{Ask}(*, *, offerte)$	special offers
Q9	$\textsc{Ask}(*, *, alloggio)$	accommodation
Q10	$\textsc{Ask}(*, *, alloggio \sqcap (mare \sqcup Europa)$	accommodation in Europe or at sea

Fig. 3. Queries performed by the suggester in the experimentation.

id.	R	T	Prc.	Rec.	# ont.	# NLP	R	T	Prc.	Rec.	# ont.	# NLP
Q1	66 (4)	82	$0.\overline{93}$	0.756	6	14	83 (6)	82	0.928	0.939	5	0
Q2	14	16	1	0.875	1	1	14	16	1	0.875	1	1
Q3	11	12	1	0.917	0	0	12	12	1	1	0	0
Q4	0	2	UNDEF	0	0	0	2	2	1	1	0	0
Q5	23 (18)	5	0.217	1	0	0	153 (148)	5	$0.0\bar{3}$	1	0	0
Q6	306 (26)	358	0.915	0.782	48	32	357 (26)	358	0.927	0.925	12	15
Q7	32	129	1	0.248	0	0	103 (7)	129	0.932	0.744	29	4
Q8	7	7	1	1	0	0	7	7	1	1	0	0
Q9	26	38	1	0.684	0	0	27	38	1	0.711	5	6
Q10	10	30	1	$0.\bar{3}$	13	7	15	30	1	0.5	8	7

Fig. 4. Experimental results, left: different ontologies, right: shared ontology

We used a list of 544 Italian stop-words. The overhearer observed a total of 160 messages on the meteorological newsgroup and 754 messages on the traveling one.

Figure 3 contains ten subscription queries posed by suggesters to the overhearer. Questions Q1–Q4 were posed on the meteorological newsgroup using the meteorological ontology. As for Q5, the same meteorological ontology was used, but on the traveling newsgroup. Finally, Q6–Q10 were posed on the traveling newsgroup with the traveling ontology. Even though the system allows for roles in the grammar, we did not used roles in the queries, because our ontologies were too simple.

The two tables of Figure 4 present the results of the queries under different assumptions. R is the number of messages that generated a notification event for the suggester (in parenthesis the number of wrong retrievals). T are the messages that were supposed to generate the notification (ground truth). Prc. and Rec. are the standard information retrieval measures [5] of precision and recall; specifically Prc.$= \frac{|R \cap T|}{|R|}$ and Rec.$= \frac{|R \cap T|}{|T|}$. The last columns of the tables represent the two main causes of missing notification to suggesters: # ont. is the number of errors due to insufficient care in the design and implementation of the ontology, while # NLP is the number of errors due to misuse of NLP-information retrieval techniques.

The numbers in the right table were obtained by assuming that overhearer and the newsgroup members share the same ontology, i.e. same word, same meaning. For the left table, we assumed this is not necessarily the case—"snow" can be either frozen water or a color (as in "snow white"). Since messages contain no reference to an ontology and our NLP capabilities are very limited, we arbitrarily derived an instance concept by assuming that this was the closest word preceding it, following the Italian syntax—snow ("neve") in snow white ("bianco come la neve") would be an instance of white ("bianco").

The results shown in Figure 4 demonstrate that our language has a significant discriminating power among the messages, and that the suggesters could express their wishes of notification without ambiguity. The only problematic case is the one of query Q5, where the structure of the meteorology ontology causes the very low value of precision. This is because some of the concepts related to geography and orography subsume part of the concepts on the traveling domain, so causing wrong notifications.

It should be noted that our aim was not that of building an information filtering system. If a robust system for interpreting natural language has to be developed, it would be necessary to employ better NLP tools. Presently only stop-words are removed; stemming techniques are also necessary, acronyms should be eliminated, given names should be handled, synonyms solved, and so on; some shallow parsing could also help. Also, the ontologies were rather small; for instance, the traveling ontology only contained geographical information of the main tourist resorts. To increase both precision and recall, richer ontologies should have been developed, but this was beyond the scope of the experimentation.

In spite of the current limitations, $\mathcal{P}_{\mathcal{ALC}}$ is significantly more powerful than using simple pattern matching expressions for single words. For instance, the patterns corresponding to Q10 are generated by the Cartesian product of the set of objects subsumed by "alloggio" (accommodation, i.e., everything from hotels to camp sites) with the union of objects subsumed by "mare" (sea) and "Europa" (any European place defined in the overhearer's ontology).

7 Some Comparisons

It is worthwhile to highlight differences and relationships between $\mathcal{PT}_{\mathcal{ALC}}$ and agent communication languages (ACLs) such as FIPA-ACL [19] (see also [7,23] in this volume) and KQML [18]. $\mathcal{PT}_{\mathcal{ALC}}$ is a logic language used by suggesters to query the log of messages kept by an overhearer in a way similar to, for instance, how database applications use SQL to interrogate a remote relational database. An ACL may be used to *transport* both the queries formulated in $\mathcal{PT}_{\mathcal{ALC}}$ and the messages exchanged by service agents. If this is the case, the chosen ACL implicitly constrains the performatives that should be expressed in $\mathcal{PT}_{\mathcal{ALC}}$. Moreover, ACLs usually allow an ontology to be explicitly referred to, so to allow the correct interpretation of the contents of a message. The ontology used by an overhearer is his own, and may or may not match (e.g., subsume) the ontologies used by services in their conversations (see the experiments in Section 6). It should be noted that nothing prevents an application from deploying multiple overhearers, with suggesters submitting $\mathcal{PT}_{\mathcal{ALC}}$ queries to those overhearers whose ontologies match theirs. The analysis of such a scenario, as well as optimizations of the interpretation of message content based on subsumption of overhearer's and services' ontologies, is left to future work.

An alternative feasible approach to the overhearer—suggester interaction is the adoption of temporal databases [9]. In this setting, the overhearer would manage an internal database updated with values extracted from the messages traveling on the channel, and the suggester would ask for notification by directly expressing temporal SQL queries (TSQL2, [27]). Some advantages can probably be gained from the point of view of performance and engineering, in particular if an application needs to permanently store the messages. On the other hand, since there is no ontological interpretation, the precision of the overhearing would be greatly reduced, affecting the quality of the suggestions. The solution with $\mathcal{PT}_{\mathcal{ALC}}$ is at a higher abstraction level and seems to be more general.

In the field of description logics, some systems are closely related to the one presented here (see [30] for an example, while [2] is an overview of temporal description logics). Often these formalisms are concerned with the modeling of concepts related to time *within* the terminological axioms, while in the overhearing architecture we are actually interested in the evolution in time of terminological expressions enriched with performatives. The latter does not mean that one cannot express temporal concepts in the ontology of the overhearer, but it does mean that these concepts do not have a specific temporal semantics assigned (e.g., like those achievable by using concrete domains [4]). Temporal concepts follow the semantics of the terminological language just like any other expressible concept.

8 Conclusions and Future Work

The work presented here originated in the field of cooperative software agents, and in particular from the overhearing architecture described in [12]. Our general focus is on interaction languages of a high abstraction level. In this work, we combined elements from temporal and description logic to provide a terse language for querying a log of messages exchanged among agents. In our opinion, its application within the overhearing architecture brings an useful enhancement to the state-of-the-art in agent collaboration, since it enables an easy detection of certain communication patterns; this may eventually lead to unsolicited help by collaborative agents. In the current setting, the overhearer has the capability to tap a channel, to use an ontology and to parse query messages in $\mathcal{PT}_{\mathcal{ALC}}$. Mentalistic interpretation of conversations—that is, building a model of the behavior of the service agents—is entirely left to the suggesters. From a performance perspective, the overhearer acts as a filter that reduces the (potentially onerous) workload of the suggesters.

In Section 5, we mentioned a few ways of enhancing the current implementation of the overhearer. We are also considering giving him some simple mental recognition capability. If, for example, an agent is asking whether someone has a pen, one is willing to think that he does not have one and it is his intention to borrow a pen and then to use it. If an agent is asking about the result of a given soccer match, one is willing to think that he does not know about the result. Once the overhearer has this extra information, the suggester must be given the chance to make queries also regarding such content. We are thinking along the lines of epistemic logics [20]. Of particular interest is the temporal epistemic multi-agent logics presented in [14]. It would be necessary to work out a semantics in the same style of $\mathcal{PT}_{\mathcal{ALC}}$, where instead of the usual valuation function, a truth definition based on description logics reasoning mechanisms is used.

In addition to improving $\mathcal{PT}_{\mathcal{ALC}}$, future work on the overhearing architecture will look at the many conceptual and computational challenges involved in understanding when to intervene in a conversation, and how to deal with unsolicited suggestions.

Acknowledgements

We thank Floriano Zini for his participation in the definition of the architecture, and the discussions on $\mathcal{PT}_{\mathcal{ALC}}$. Special thanks are due to ITC-IRST, to the University of Trento and to Prof. Fausto Giunchiglia for supporting the extended visit of Marco Aiello that lead to this research. Finally, we would like to thank the anonymous referees for the helpful comments.

References

1. Agent Oriented Software. *JACK Intelligent Agents User Manual ver. 3.0*, 2001. `http://www.jackagents.com`.
2. A. Artale and E. Franconi. A survey of temporal extensions of description logics. volume 30 (1–4). Kluwer Academic Publishers, 2001.
3. F. Baader, H. Burckert, J. Heinsohn, B. Hollunder, J. Muller, B. Nebel, W. Nutt, and H. Profitlich. Terminological knowledge representation: a proposal for a terminological logic. Technical report, DFKI, Saarbrucken, 1992.
4. F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In *IJCAI*, pages 452–457, 1991.
5. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1999.
6. M. V. Belmonte, C. Conejo, J. L. Pérez-de-la-Cruz, and F. Triguero. A Stable and Feasible Payoff Division for Coalition Formation in a Class of Task Oriented Domains. In J. J. Meyer and M. Tambe, editors, *Intelligent Agents VIII. Agents Theories, Architectures, and Languages (ATAL) — 8th International Workshop*, Seattle, Washington, USA, August 2001. Springer-Verlag. In this volume.
7. F. Bergenti and A. Poggi. LEAP: A FIPA Platform for Handheld and Mobile Devices. In J. J. Meyer and M. Tambe, editors, *Agents Theories, Architectures, and Languages (ATAL) — 8th International Workshop*, Seattle, Washington, USA, August 2001. Springer-Verlag. In this volume.
8. E. Blanzieri, P. Giorgini, P. Massa, and S. Recla. Implicit Culture for Multi-agent Interaction Support. In *Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS 2001)*, Trento, Italy, September 2001.
9. M. Boehlen, J. Chomicki, R. Snodgrass, and D. Toman. Querying TSQL2 databases with temporal logic. In *5th International Conference on Extending Database Technology (EDBT)*, Avignon, 1996.
10. A. Borgida and P. Patel-Schneider. A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. *Journal of Artificial Intelligence Research*, 1:277–308, 1994.

11. F. Brandt and G. Weiss. Antisocial Agents and Vickrey Auctions. In J. J. Meyer and M. Tambe, editors, *Intelligent Agents VIII. Agents Theories, Architectures, and Languages (ATAL) — 8th International Workshop*, Seattle, Washington, USA, August 2001. Springer-Verlag. In this volume.
12. P. Busetta, L. Serafini, D. Singh, and F. Zini. Extending Multi-Agent Cooperation by Overhearing. In *Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS 2001)*, Trento, Italy, September 2001. ITC-irst Technical Report 0101-01.
13. P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg. An open agent architecture. In *Proceedings of the AAAI Spring Simposium on Software Agents*, pages 1–8. AAAI Press, 1994.
14. C. Dixon, M. Fisher, and M. Wooldridge. Resolution for Temporal Logics of Knowledge. *Journal of Logic and Computation*, 8(3):345–372, 1998.
15. J. Doran, S. Franklin, N. Jennings, and T. Norman. On Cooperation in Multi-Agent Systems. *The Knowledge Engineering Review*, 12(3), 1997.
16. P. Faratin, M. Klein, H. Sayama, and Y. Bar-Yam. Simple negotiation agents in complex games. In J. J. Meyer and M. Tambe, editors, *Intelligent Agents VIII. Agents Theories, Architectures, and Languages (ATAL) — 8th International Workshop*, Seattle, Washington, USA, August 2001. Springer-Verlag. In this volume.
17. S. S. Fatima, M. Wooldridge, and N. R. Jennings. Optimal negotiation strategies for agents with incomplete information. In J. J. Meyer and M. Tambe, editors, *Intelligent Agents VIII. Agents Theories, Architectures, and Languages (ATAL) — 8th International Workshop*, Seattle, Washington, USA, August 2001. Springer-Verlag. In this volume.
18. T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In Jeff Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997.
19. Foundation for Intelligent Physical Agents. *FIPA ACL Message Structure Specification*, 2000. Document number 00061D. `http://www.fipa.org/`.
20. J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.
21. I. Horrocks. Using an expressive description logic: FaCT or fiction? In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647. Morgan Kaufmann Publishers, San Francisco, California, 1998.
22. M. Klusch, editor. *Intelligent Information Systems*. Springer-Verlag, 1999.
23. M. Laukkanen, S. Tarkoma, and J. Leinonen. FIPA-OS Agent for Small-footprint Devices. In J. J. Meyer and M. Tambe, editors, *Intelligent Agents VIII. Agents Theories, Architectures, and Languages (ATAL) — 8th International Workshop*, Seattle, Washington, USA, August 2001. Springer-Verlag. In this volume.
24. Metamata and Sun Microsystems. *JavaCC Documentation, ver. 2.0*, 2001. `http://www.metamata.com/javacc/`.
25. T. Oates, M. Prasad, and V. Lesser. Cooperative Information Gathering: A Distributed Problem Solving Approach. *IEEE Proc. on Software Engineering*, 144(1), 1997.
26. F. Sadri, F. Toni, and P. Torroni. Dialogues for negotiation: agent varieties and dialogue sequences. In J. J. Meyer and M. Tambe, editors, *Intelligent Agents VIII. Agents Theories, Architectures, and Languages (ATAL) — 8th International Workshop*, Seattle, Washington, USA, August 2001. Springer-Verlag. In this volume.
27. B. Salzberg and V. J. Tsotras. Comparison of Access Methods for Time-Evolving Data. *Computing Surveys*, 31(2):158–221, 1999.
28. M. Schmidt-Schau and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.
29. Y. Venema. Temporal logics. In L. Goble, editor, *Blackwell's Guide to Philosophical Logic*. Basil Blackwell Publishers, Cambridge, MA, To appear.
30. F. Wolter and M. Zakharyaschev. Temporalizing description logics. In *FroCoS'98*, Amsterdam, 1998. Kluwer.

Agent Dialogues with Conflicting Preferences

Leila Amgoud[1] and Simon Parsons[2]

[1] IRIT-UPS, 118 route de Narbonne 31062
Toulouse Cedex, France.
amgoud@irit.fr

[2] Department of Computer Science, University of Liverpool
Liverpool L69 7ZF, United Kingdom.
s.d.parsons@csc.liv.ac.uk

Abstract. An increasing number of software applications are being conceived, designed, and implemented using the notion of autonomous agents. These agents need to communicate in order to resolve differences of opinion and conflicts of interests, some of which result from differences in preferences. Hence the agents need the ability to engage in dialogues that can handle these communications while taking the differences in preferences into account. In this paper we present a general framework of dialogue where several agents can fulfill the above goals.

1 Introduction

An increasing number of software applications are being conceived, designed, and implemented using the notion of autonomous agents. These applications vary from email filtering, through electronic commerce, to large industrial applications. In all of these disparate cases, however, the notion of autonomy is used to denote the fact that the software has the ability to decide for itself which goals it should adopt and how these goals should be achieved. In most agent applications, the autonomous components need to interact. They need to communicate in order to resolve differences of opinion and conflicts of interest, work together to resolve dilemmas or find proofs, or simply to inform each other of pertinent facts. Often these needs require agents to engage in dialogues.

As a result of this requirement, there has been much work[1] on providing agents with the ability to hold dialogues. Typically these focus on one type of dialogue, either negotiation [14,18,19], where agents try to reach agreement on the division of some set of scarce resources, or persuasion [5,7,10,12,16,20], where one agent is trying to change the opinion of another. In previous work [2,3] we have presented a set of illocutions which can capture a number of different types of dialogue, An important limitation of much of this existing work is the fact that it either doesn't take into account the preferences of the agents (eg [10,12,16], or, as in [2,3], it assumes all agents have the same preferences. Another limitation

[1] This is meant to be representative rather than an exhaustive survey. In particular, it omits the literature on natural language dialogues because we are interested in the much more restricted task of handling simple dialogues between software agents.

is that while the way that the argumentation is handled is considered in great detail, much less attention is often paid to the way that arguments fit into dialogues, the way that these dialogues are constructed, and the overall context in which the dialogue takes place.

In this paper, we take some steps towards tackling these limitations. Firstly, we extend the argumentation system of [2,3] to give the agents the ability to engage in dialogues while taking different sets of preferences into account. Secondly we show how this extended system fits into the wider context of agent dialogues by presenting a general framework which captures many of the essential features of such dialogues, and provide an instantiation of this framework which brings together elements of our previous work on this subject.

2 Arguing with Conflicting Preferences

Argumentation is an approach to handle reasoning about inconsistent information, based on the justification of plausible conclusions by arguments. Broadly speaking, any conclusion is initially entertained so long as an argument can be constructed in favour of it. This results in a set of arguments which, in general, conflict because they disagree about the truth of certain propositions. From this set some subset of conclusions are identified as acceptable, based on the relationships between the conflicting arguments.

In previous work [2,3] we have described in detail a particular formal system for argumentation and shown how it can be used to underpin dialogues between agents under the assumption that all agents have the same set of preferences. Here we extend this system to take account of different sets of preferences, adopting the approach we suggested in [4]. We take the different preferences to be expressed as different preorderings over a set of propositions representing beliefs, and consider that different preferences arise because the propositions are considered from the viewpoint of different contexts. Thus the different preference orderings can be considered as *contextual preferences*, and these can change as agents update their knowledge.

We start with a single agent which has a possibly inconsistent knowledge base Σ with no deductive closure. We assume Σ contains formulae of a propositional language $\mathcal{R}$. $\vdash$ stands for classical inference and $\equiv$ for logical equivalence. The agent's beliefs are context-specific, and we denote this set of contexts by $\mathcal{C} = \{c_1, \ldots, c_n\}$. We assume that there is an order over these contexts, $\sqsupset$, so that for $c_1, c_2 \in \mathcal{C}$, $c_1 \sqsupset c_2$ means that any proposition in context c_1 is preferred over any proposition in context c_2. In addition to the ordering over contexts, there is also a set of preorderings $\gg_i, \ldots, \gg_n$ which give the preference over propositions within every context, $\gg_i$ giving the preferences in context c_i. We have:

Definition 1. *An* argument *is a pair $A = (H, h)$ where h is a formula of $\mathcal{R}$ and H a subset of Σ such that:*

1. *H is consistent;*
2. *$H \vdash h$; and*
3. *H is minimal, so no subset of H satisfying both 1. and 2. exists.*

H is called the support *of A, written H = Support(A), and h is the* conclusion *of A, written h = Conclusion(A).*

Since Σ is inconsistent, $\mathcal{A}(\Sigma)$, the set of all arguments which can be made from Σ, will contain arguments which undercut each other:

Definition 2. *Let A_1 and A_2 be two arguments of $\mathcal{A}(\Sigma)$. A_1* undercuts *A_2 iff $\exists h \in Support(A_2)$ such that $h \equiv \neg Conclusion(A_1)$.*

In other words, an argument is undercut iff there exists an argument for the negation of an element of its support.

Each preordering on the set Σ can be used to define a preordering on the set of arguments $\mathcal{A}(\Sigma)$. We can thus define a preference relation $Pref_i$ over a context c_i based on the appropriate preordering $\gg_i$. In [1] several preference relations between arguments were proposed. Some of these assume that each $\gg_i$ is a partial preordering and others take it to be a total ordering. In this paper we adopt a preference relation which takes $\gg_i$ to be a total preordering. This is equivalent to considering the knowledge base to be stratified into non-overlapping sets $\Sigma_1, \ldots, \Sigma_n$ such that facts in Σ_i are all equally preferred and are more preferred than those in Σ_j where $j > i$. The preference level of a nonempty subset H of Σ, $level(H)$, is the number of the highest numbered layer which has a member in H.

Definition 3. *Let A_1 and A_2 be two arguments in $\mathcal{A}(\Sigma)$. A_1 is* preferred *to A_2 according to $Pref_i$ iff $level(Support(A_1)) \leq level(Support(A_2))$.*

We denote by $Pref_1$, ..., $Pref_n$ the different preference relations between arguments induced respectively from the preorderings $\gg_1$, ..., $\gg_n$. Note that since the preorderings on Σ may be conflicting, the preorderings on $\mathcal{A}(\Sigma)$ also may be conflicting, so that for two arguments A_1 and A_2, A_1 may be preferred to A_2 in a context c_i and A_2 may be preferred to A_1 in a context c_j such that $i \neq j$. We can now formally define the argumentation system we will use. We will denote such a system by CPAS:

Definition 4. *An* argumentation system based on contextual preferences *is a tuple $\langle \mathcal{A}(\Sigma)$, Undercut, $\mathcal{C}$, $\sqsupset$, $\gg_1$, ..., $\gg_n \rangle$ such that:*

- *$\mathcal{A}(\Sigma)$ is the set of arguments built from Σ;*
- *Undercut is a binary relation identifying which arguments undercut which other arguments, Undercut $\subseteq \mathcal{A}(\Sigma) \times \mathcal{A}(\Sigma)$;*
- *$\mathcal{C}$ is a set of contexts $\{c_1, \ldots, c_n\}$;*
- *$\sqsupset$ is a (total or partial) preordering on $\mathcal{C} \times \mathcal{C}$; and*
- *$\gg_i$ is a (partial or total) preordering on $\Sigma \times \Sigma$ in the context c_i.*

The preference orderings $Pref_1$, ..., $Pref_n$ make it possible to distinguish different types of relations between arguments based on the way in which a set of arguments mutually undercut one another. Broadly speaking, an argument defends itself if it is stronger (in the sense of being based in a preferred context) than those arguments which seek to undercut it, and a set of arguments can defend a lone argument by undercutting all those arguments which the lone argument cannot defend itself against:

Definition 5. *Let A_1, A_2 be two arguments of $\mathcal{A}(\Sigma)$.*

- *If A_2 undercuts A_1 then A_1* defends itself *against A_2 iff:*
  1. *$\exists c_i \in \mathcal{C}$ such that A_1 $Pref_i$ A_2 and*
  2. *$\forall c_j$ such that A_2 $Pref_j$ A_1 then $c_i \sqsupset c_j$.*

  Otherwise, A_1 does not defend itself.
- *A set of arguments $\mathcal{S}$* defends *A iff $\forall B$ such that B undercuts A and A does not defend itself against B then $\exists C \in \mathcal{S}$ such that C undercuts B and B does not defend itself against C.*

Henceforth, $C_{Undercut,\sqsupset}$ will refer to all non-undercut arguments and arguments defending themselves against all their undercutting arguments. In [4], it was shown that the set $\underline{\mathcal{S}}$ of acceptable arguments of the argumentation system $\langle \mathcal{A}(\Sigma),\ Undercut,\ \mathcal{C},\ \sqsupset,\ \gg_1,\ \ldots,\ \gg_n \rangle$ is the least fix-point of a function $\mathcal{F}$:

$$\mathcal{S} \subseteq \mathcal{A}(\Sigma)$$
$$\mathcal{F}(\mathcal{S}) = \{A \in \mathcal{A}(\Sigma) | A \text{ is defended by } \mathcal{S}\}$$

which then leads us to be able to define those arguments which are acceptable in the sense of either defending themselves or being defended:

Definition 6. *The set of* acceptable *arguments for an argumentation system $\langle \mathcal{A}(\Sigma),\ Undercut,\ \mathcal{C},\ \sqsupset,\ \gg_1,\ \ldots,\ \gg_n \rangle$ is:*

$$\underline{\mathcal{S}} = \bigcup \mathcal{F}_{i \geq 0}(\emptyset) = C_{Undercut,\sqsupset} \cup \left[\bigcup \mathcal{F}_{i \geq 1}(C_{Undercut,\sqsupset}) \right]$$

An argument is acceptable if it is a member of the acceptable set.

In practice we don't need to calculate all the acceptable arguments from Σ in order to know whether or not a given argument is acceptable [2].

We can use this argumentation system to straightforwardly extend our previous work on argumentation-based dialogues [2,3] to deal with agents that have different preferences.

3 A Dialogue System

In this section we show how the system of argumentation introduced above may be used in inter-agent dialogues. We start from an abstract notion of a dialogue system which captures the basic elements required for such dialogues.

3.1 A General Framework

One basic element in a dialogue system is the set of agents involved in a dialogue. Dialogues are often considered to take place between two agents, but here we deal with dialogues between arbitrary numbers of agents. Each agent has a name, a role in the dialogue and a knowledge base. The knowledge-base may contain information about other agents and we assume it is equipped with a (partial or total) preordering representing the preferences of the agent. The knowledge-base is written in some logic, and different agents can use different logics.

Following Hamblin [9] and Mackenzie [10], we suppose that the dialogue takes place using *commitment stores* that hold all the statements to which an agent has committed during the dialogue (broadly speaking all the propositions it has agreed are true). The different commitment stores are empty at the beginning of a dialogue and the rules for carrying out dialogues define how the commitment stores are updated. Thus the union of all the commitment stores can be viewed as the state of the dialogue at a particular time. Furthermore, the commitment stores provide additional knowledge, over and above that in an agent's knowledge-base, which an agent can use as the basis for its utterances.

We bring these ideas together in the notion of a dialogical agent:

Definition 7. *A* dialogical agent *is a tuple* $\langle A_i, Role_{A_i}, \mathcal{L}_{A_i}, \Sigma_{A_i}, \gg_{A_i}, CS_{A_i}\rangle$ *where:*

- A_i *is the name of the agent;*
- $Role_{A_i}$ *denotes the role of agent* A_i*;*
- $\mathcal{L}_{A_i}$ *denotes the logic used by agent* A_i*;*
- Σ_{A_i} *is the knowledge base available to the agent* A_i*;*
- $\gg_{A_i}$ *represents the preference order over* Σ_{A_i}*; and*
- CS_{A_i} *stands for the commitment store of agent* A_i*.*

We denote a single dialogical agent by $\mathcal{A}_i$ *and define a* set of dialogical agents *as* $\mathsf{A} = \bigcup_i \{\mathcal{A}_i\}$.

The role of an agent may affect the kind of logic used by that agent. In dialogues where a judge determines the outcome, some agents may use classical logic whereas the judge requires a more sophisticated non-classical logic in order to handle the contradictory arguments presented. In some applications, the strength of arguments are determined by roles. As discussed in [17], the hierarchy of a company may be reflected in the weight accorded to arguments made by agents playing certain roles. Hence, the set of agents may be equipped with a (partial or total) preordering $\succ$ to capture these differences in the strength of argument that an agent derives from its role.

Now, a dialogue may be viewed as a sequence of speech acts made by agents:

Definition 8.

A move *in a dialogue is a tuple:* $M = \langle S, H, Act\rangle$. S *is the agent providing the act, written* $S = Speaker(M)$. H *is the agent or set of agents to whom the act is addressed, written* $H = Hearer(M)$. *When the act is addressed to all the agents, we denote* A_Ω. $Act = Act(M)$ *is the act itself.*

A dialogue *is a non-empty sequence of moves* $M_1, \ldots, M_p$ *such that:*

- $Speaker(M_j) \neq Hearer(M_j)$
- $CS_{A_i}(0) = \emptyset$, $\forall i = 1, \ldots n$. *Note that* $CS_{A_i}(0)$ *is the commitment store of agent* A_i *at step 0.*
- *For any two moves* M_j, M_k, *if* $j \neq k$ *then* $Act(M_j) \neq Act(M_k)$
- *For any* $j < p$: $Speaker(M_j) \neq Speaker(M_{j+1})$.

The first condition prevents an agent from addressing a move to itself. The second says that commitment stores are empty at the beginning of the dialogue. The third prevents an agent from repeating an act already provided by another agent or by the agent itself (in other words, for example, it prevents the an agent repeatedly asking the same question). This guarantees non circular dialogues. The last condition prevents an agent from providing several moves at the same time and guarantees that there are no monologues.

Note the formal distinction between dialogue moves and dialogue acts. An act is a locution ($assert(p)$, $challenge(q)$ and so on), and makes up part of a move along with the agent generating the act and the agent receiving the act. When there is no chance of confusing the terms (or, as in most situations, it is possible to correctly use either), we will use "act" and "move" interchangeably and we frequently use M to denote a set of acts.

Agents are not free to make any move at any time—their moves are governed by a protocol, a set of rules governing the high-level behavior of interacting agents. A given protocol specifies the kinds of moves or acts the agents can make (assertions, requests, and so on) at any point in the dialogue.

Definition 9. *A* dialogue protocol *is defined as a function* $\pi : \mathsf{M} \mapsto 2^{\mathsf{M}}$*, where* M *is a set of dialogue acts.*

A dialogue protocol specifies the rules for interactions between agents and the different replies that are possible after a given move. In general there will be several such moves possible, and the exact way that an agent responds is the result of the agent's strategy. We can therefore think of a given strategy for an agent as being a function from the set of moves identified by the protocol to a single move which is then uttered by an agent:

Definition 10. *A* dialogue strategy *is defined as a function* $\mathcal{S} : 2^{\mathsf{M}} \mapsto \mathsf{M}$*, where* M *is a set of dialogue acts, such that for* $T \subseteq \mathsf{M}$*,* $\mathcal{S}(T) \in T$*.*

Given these definitions, we can define a dialogue system. Such a system will consist of a set of dialogical agents with an ordering over them, a protocol (which includes a set of speech acts), a set of strategies (one strategy for each agent), and a dialogue (which can be thought of as a tape recording of everything that has been said in the dialogue). Formally:

Definition 11. *A* dialogue system *is a tuple* $\langle \mathsf{A}, \succ, \mathsf{M}, \pi, \mathsf{S}, \mathcal{D} \rangle$ *where* A *is a set of dialogical agents,* $\succ$ *is a preordering over the elements of* A*,* M *is a set of acts,* π *is a dialogue protocol,* S *is a set of strategies, and* $\mathcal{D}$ *is a dialogue.*

In the rest of this section we will illustrate the idea of a dialogue system by instantiating each of these elements with a specific example.

3.2 The Set of Agents

We consider a set of agents $\mathsf{A} = \{A_1, \ldots, A_n\}$, $n \geq 2$, where A_i is the name of the ith agent. For now we make no explicit use of the agent roles, noting only that, as in [17], roles can play a part in the formation of the preference order $\succ$ over the agents. Note that we only consider dialogues between at least two agents. As in [14], we assume that each agent has a set of beliefs, B, a set of desires, D, and a set of intentions, I. However, for the moment we deal with propositional knowledge and take each agent to use only classical propositional logic, modelling beliefs, desires and intentions by partitioning the knowledge base of each agent appropriately. In particular, we take the *basic knowledge base* of an agent A_i to be:

$$\Sigma^B_{A_i} = B_{A_i} \cup D_{A_i} \cup I_{A_i}$$

and assume that we know that a particular proposition p is, for example, one of A_i's intentions because it resides in I_{A_i}, not because it is explicitly denoted as such. We work under this restriction for notational simplicity, bearing in mind that the problem of how to deal with first-order argumentation in which beliefs, desires and intentions are explicitly denoted is considered in depth in [14].

In addition, the propositional language can usefully be extended to represent the type of information exchanged between agents in negotiation. As discussed in [17], negotiations often involve trade-offs with one agent accepting a request from a second agent provided that the second accepts its request. For example: "If you let me use your laptop, I'll let use my printer". To make it easier to represent this kind of information a new connective $\Rightarrow$ was introduced in [3]. Thus we have a new language $\mathcal{L}$ which contains propositional formulae and formulae $p \Rightarrow q$ such that p and q are propositional formulae. Note that for this instantiation of our general framework, all agents are assumed to have the same logic. As real multi-agent systems employ different logics, in [11] a formalism allowing reasoning with different rules of inference has been defined.

As described above, each agent A_i uses a commitment store CS_{A_i} in order to keep track of its dialogue commitments. At the beginning of the dialogue the different commitment stores are empty, and agents are assumed to have free access to any information stored in them. Since agents have to exchange two kinds of information—knowledge and preferences—the commitment stores will have two parts. The first part, denoted by $CS.Pref_{A_i}$, will contain preferences, and the second part, denoted by $CS.Kb_{A_i}$, will contain knowledge:

$$CS_{A_i} = CS.Pref_{A_i} \cup CS.Kb_{A_i}$$

Agent A_i knows everything in the commitment stores of all the agents in the dialogue, $\cup^n_{j=1} CS.Kb_{A_j}$, and also has some information about the beliefs of each agent, $\cup_{j \neq i} \Sigma^{B'}_{A_j}$ with $\Sigma^{B'}_{A_j} \subseteq \Sigma^B_{A_j}$. Thus the overall knowledge available to A_i is:

$$\Sigma_{A_i} = \Sigma^B_{A_i} \cup [\cup_{j \neq i} \Sigma^{B'}_{A_j}] \cup [\cup^n_{j=1} CS.Kb_{A_j}]$$

Each agent also has preferences over its knowledge base, and so is equipped with a (total or partial) preordering on $\Sigma^B_{A_i}$, denoted by $\gg^B_{A_i}$. These preferences are given as pairs (a, b) which denotes a is preferred to b.

A_i also knows some of the preferences of other agents A_j. These preferences, a subset of those in $\gg^B_{A_j}$, will be denoted by $\gg^{B'}_{A_j}$. A_i also knows the preferences that the other agents have stated, and are thus in commitment stores. Hence it knows $\gg_{j=1,n} CS.Pref_{A_j}$. So the overall set of preferences that A_i knows about are:

$$\gg_{A_i} = \gg_{A_i}^{B} \cup [\cup_{j \neq i} \gg_{A_j}^{B'}] \cup [\cup_{j=1,n} CS.Pref_{A_j}]$$

Each agent is equipped with an argumentation framework of the kind discussed above as its inference mechanism[2]. Using Σ_{A_i}, A_i can build arguments concerning beliefs, desires and intentions as discussed above. We adopt the system from Section 2 treating agents as contexts, so the set of contexts $\mathcal{C}$ is replaced by the set of agents A and the preordering $\sqsupset$ on the contexts replaced by the preordering $\succ$ on the set of agents. Thus each agent A_i will use the argumentation system:

$$\langle \mathcal{A}(\Sigma_{A_i}), \mathit{Undercut}, \mathsf{A}, \succ, (\gg_{A_1}^{B'} \cup CS.Pref_{A_1}), \ldots, (\gg_{A_n}^{B'} \cup CS.Pref_{A_n}) \rangle$$

The argumentation frameworks are used to help the agents to maintain the coherence of their beliefs, and ensure that they only assert justified arguments. In this sense the argumentation systems help to operationalize the *rationality* of the agents, and, as in [2], this in turn ensures that if dialogues end, they end with agents agreeing on arguments which are acceptable to all agents in the dialogue.

3.3 Dialogue Acts and *Protocol*

A set of dialogue acts and a protocol together these define the full set of legal moves available to an agent at any given time, and the mapping from one move in dialogue to the set of possible next moves. In this paper we combine these, as in [3], by giving each act an associated set of rules for using it—update rules, dialogue rules and rationality rules. The update rules simply say how the commitment stores of all the agents are updated as a result of the act. The rationality rules and the dialogue rules are more complex. The rationality rules are preconditions for an act, for instance saying that the act can only be made if the agent can build a certain argument. The dialogue rules define the acts which can be used to respond to a given act. Thus the rationality and dialogue rules together identify the set of possible next moves, and hence make up a protocol. Given a particular move, the dialogue rules for that move identify a set of possible replies and the rationality rules for these possible replies then identify which can be legally used.

In the following descriptions, we suppose that agent A_i addresses an act to the other agents. The CPAS is therefore:

$$\langle \mathcal{A}(\Sigma_{A_i}), \mathit{Undercut}, \mathsf{A}, \succ, (\gg_{A_1}^{B'} \cup CS.Pref_{A_1}), \ldots, (\gg_{A_n}^{B'} \cup CS.Pref_{A_n}) \rangle.$$

Basic Dialogue Acts

assert(p) where p is any formula in $\mathcal{L}$. This allows the exchange of information, such as "the weather is beautiful" or "It is my intention to hang a picture".

[2] We view the meta-level inference provided by the argumentation system, along with the underlying propositional logic to be the $\mathcal{L}_{A_i}$ of an agent.

rationality The agent uses the CPAS to check if there is an acceptable argument A such that $p = Conclusion(A)$.

dialogue The other agents can respond with:

1. $accept(p)$,
2. $assert(\neg p)$,
3. $challenge(p)$.

update $CS.Kb_{A_i}(t) = CS.Kb_{A_i}(t-1) \cup \{p\}$ and $CS.Kb_{A_j}(t) = CS.Kb_{A_j}(t-1)$, $\forall j \neq i$.

This assertion is added to the CS of the agent making the assertion. Note that an agent A_j can only make a response if the rationality rule for that response is satisfied. Thus it can only respond to $assert(p)$ with $assert(\neg p)$ if it has an acceptable argument for $\neg p$. $assert(S)$ where S is a set of formulae in $\mathcal{L}$ representing the support of an argument, is handled in a similar manner [3].

An agent is also allowed to present its preferences, requiring a new locution in addition to those in [3]:

$prefer((a_1, b_1), \ldots, (a_n, b_n))$ where a_i, b_i are formulae in $\mathcal{L}$.

rationality There is no rationality condition.

dialogue The other agents can play:

1. $prefer((a'_1, b'_1), \ldots, (a'_j, b'_j))$,
2. $assert(S)$,
3. $question(q)$,
4. $request(q)$, where $q = b_i$,
5. $promise(x \Rightarrow a_i)$.

update $CS.Pref_{A_i}(t) = CS.Pref_{A_i}(t-1) \cup \{(a_1, b_1), \ldots, (a_j, b_j)\}$ and $CS.Pref_{A_j}(t) = CS.Pref_{A_j}(t-1)$, $\forall j \neq i$.

Informally, this means that the responding agent can present its preferences, give an argument, ask a question, request something not preferred by the original agent, or simply promise something preferred by the other agent in exchange for another element. The next two moves allow an agent to ask questions.

$challenge(p)$ where p is a formula in $\mathcal{L}$.

rationality There is no rationality condition.

dialogue The other agents can only $assert(S)$ where S is the support of the argument (S, p), or S is the support of the argument (S, h) such that p belongs to S and h is one of A_i's intentions.

update $CS.Kb_{A_i}(t) = CS.Kb_{A_i}(t-1)$ and $CS.Kb_{A_j}(t) = CS.Kb_{A_j}(t-1), \forall j \neq i$.

$question(p)$ [3] allows A_i to ask if p is the case. The other agents can answer either affirmatively (if they can show it to be the case) or negatively, if they can show it is not the case, or by asking another question, or by making a request.

Negotiation Acts

The following acts are negotiation specific—while not strictly necessary for negotiation, they make it easier to capture some of the statements we wish our agents to make.

$request(p)$ where p is any formula in $\mathcal{L}$.

> **rationality** A_i uses the CPAS to identify a p in some $\Sigma_{A_j}^{B'}$ such that $p \in H$ and (H, h) is an acceptable argument for one of A_i's intentions.
> **dialogue** The agent A_j can:
> 1. $accept(p)$,
> 2. $refuse(p)$,
> 3. $challenge(p)$,
> 4. $promise(q \Rightarrow p)$.
>
> **update** $CS.Kb_{A_i}(t) = CS.Kb_{A_i}(t-1)$ and $CS.Kb_{A_j}(t) = CS.Kb_{A_j}(t-1) \cup \{p\}$.

A request is stored in the CS of the receiving agent because, if accepted, it becomes a commitment on that agent. A *request* locution is invoked when an agent cannot, or prefers not to, achieve its intentions alone. The proposition requested differs from an asserted proposition in that it cannot be proved true or false—the decision on whether to accept it or not hinges upon the relation it has to other agents' intentions (see below).

$promise(p \Rightarrow q)$ where p and q are formulae in $\mathcal{L}$.

> **rationality** A_i uses the CPAS to identify a p in some $\Sigma_{A_j}^{B'}$ such that $p \in H$ and (H, h) is an acceptable argument for one of A_i's intentions, and to check that there is no acceptable argument (H', h') for one of its intentions h' such that $q \in H'$.
> **dialogue** The agent A_j can:
> 1. $accept(p \Rightarrow q)$,
> 2. $refuse(p \Rightarrow q)$,
> 3. $promise(s \Rightarrow p)$,
> 4. $challenge(p)$,
> 5. $prefer((x, q))$.
>
> **update** $CS.Kb_{A_i}(t) = CS.Kb_{A_i}(t-1) \cup \{q\}$ and $CS.Kb_{A_j}(t) = CS.Kb_{A_j}(t-1) \cup \{p\}$.

Broadly speaking, an agent will make a promise when it needs to request something, p, from another, and has something it does not need (because the thing is not needed to achieve any intentions), q, which it can offer in return. In replying to a promise, an agent can accept, refuse, question why the requested thing is required, or suggest an alternative trade (A_j replying with $s \Rightarrow p$ is equivalent to A_i retracting its initial promise and replacing it with $p \Rightarrow s$).

Responding Acts

The following are acts which are made in response to requests and assertions.

accept(p) where p is a formula in $\mathcal{L}$. After an assertion or request, an agent can respond with an explicit acceptance.

rationality In response to an assertion, A_i uses its CPAS to check if there is an acceptable argument for p. If so, the move can be played. In response to a request, A_i has to check that there is no acceptable argument (H, h) for one of its intentions h, such that $p \in H$. In other words, it is only possible to accept a request if it doesn't invalidate the supporting argument for one of its intentions[3].
dialogue The other agents can make any move except $refuse$.
update $CS.Kb_{A_i}(t) = CS.Kb_{A_i}(t-1) \cup \{p\}$ and $CS.Kb_{A_j}(t) = CS.Kb_{A_j}(t-1), \forall j \neq i$.

Note that in case of a response to a request, p is already in the commitment store of the agent. An agent can also accept a set of formulae S, *accept*(S), dealing with each member s of S as for *accept*(s). An agent can also accept a promise, using *accept*($p \Rightarrow q$) [3].

$refuse(p)$ where p is any formula in $\mathcal{L}$.

rationality A_i uses the CPAS to check if there is an acceptable argument (H, h) for one of its intentions h such that $p \in H$.
dialogue The other agents can make any move except $refuse$.
update $CS.Kb_{A_i}(t) = CS.Kb_{A_i}(t-1) \backslash \{p\}$ and $CS.Kb_{A_j}(t) = CS.Kb_{A_j}(t-1), \forall j \neq i$.

Thus A_i will refuse requests which are necessary to achieve its intentions. There is also a $refuse$ for promises [3] which reverses the effect of the previous locution on the commitment stores. As discussed in [2,3], these acts are sufficient to capture many types of dialogue, making the system more general than others which concentrate on just persuasion or negotiation. In addition, the acts closely relate the moves to the argumentation performed by an agent and hence, through the rationality conditions, to the information it has available to it.

This complete set of dialogue acts we denote $\mathcal{M}'_p$ after those developed in [2,3], both of which are subsets of $\mathcal{M}'_p$. $\mathcal{M}_{\mathcal{DC}}$ from [2] contains *assert*, *accept*, *challenge* and *question*, and $\mathcal{M}'$ from [3] contains all the acts from $\mathcal{M}_{\mathcal{DC}}$ along with *request*, *promise* and $refuse$.

3.4 Dialogue Strategy

Once the protocol has determined the set of legal moves available to an agent, the strategy selects one move from that set. For the moves we have here, we can identify a number of strategies which reflect broad classes of agent types:

[3] As in [14], an argument for an intention is essentially a plan for achieving it, so allowing p would invalidate this plan.

- Agreeable: *accept* whenever possible.
- Disagreeable: only *accept* when no reason not to.
- Open-minded: only *challenge* when necessary.
- Argumentative: *challenge* whenever possible.
- Elephant's Child [8]: *question* whenever possible.

Although we have termed these "strategies" each is only a partial definition of $\mathcal{S}$—a full definition would have to take into account the nature of the previous move and hence the overall protocol for the dialogue. For example, the agreeable strategy defines how to respond to $assert(p)$ when the agent does not have an argument against p. If the agent does have an argument, then a complete strategy would have to choose between $assert(\neg p)$ and $challenge(p)$. More work is required to define such complete strategies, and to work out the interaction between the strategies and rationality rules and their impact, and one approach to this is explored in [15].

The choice of strategy can have a big impact on the properties of the dialogue, especially on those related to length and termination. For example, as discussed below agents using the Elephant's Child strategy can cause a dialogue to spiral into an endless round of *questions*, while an argumentative agent has the potential to prolong any dialogue by constantly requiring other agents to assert arguments unnecessarily. Some initial work on this question is reported in [15] showing that termination is guaranteed in some kinds of dialogue using a subset of the dialogue moves discussed here.

3.5 Properties

Consider dialogues which start with one agent asserting some proposition (a typical start for dialogues which involve one agent trying to persuade another of the truth of a proposition). For such *assertion-led* dialogues we have [2][4] the following:

Theorem 12. *Given two agents P and C, equipped with argumentation systems AS_P and AS_C respectively, which hold an assertion-led dialogue using the set of dialogue acts $\mathcal{M}_{\mathcal{DC}}$ in which P moves first, then if S is the set of all arguments which the game can possibly generate,*

- *$\forall x \in S$, x is in the set of acceptable arguments of either AS_P or AS_C;*
- *If $x \in S$ is in the set of acceptable arguments of AS_P, it is in the set of acceptable arguments of AS_C.*

This theorem follows directly from the definition of the rationality conditions of the set of dialogue acts. As a result, it extends directly to $\mathcal{M}'$and $\mathcal{M}'_p$:

[4] In [2], the wording of the theorem is in terms of "justified arguments" which under the proof theory we use for constructing arguments (see [2] for details) are just the same as acceptable arguments, and the dialogues, though assertion-led, are not explicitly stated as such.

Theorem 13. *Given two agents P and C, equipped with argumentation systems AS_P and AS_C respectively, which hold an assertion-led dialogue using the set of dialogue acts $\mathcal{M}'$ or $\mathcal{M}'_p$ in which P moves first, then if S is the set of all arguments which the game can possibly generate,*

- *$\forall x \in S$, x is in the set of acceptable arguments of either AS_P or AS_C;*
- *If $x \in S$ is in the set of acceptable arguments of AS_P, it is in the set of acceptable arguments of AS_C.*

These results give us some guarantees about the soundness of the dialogues (they only involve arguments that at least one agent finds acceptable) and about the way the dialogue can terminate (if the dialogue ends with an argument being acceptable to P, then it is also acceptable to C, so that P can be considered to have "persuaded" C). They can readily be extended to multi-party dialogues.

We can also give results relating to termination (supplementing those in [15]). For instance:

Theorem 14. *Given two agents P and C that both use the Elephant's Child strategy, any dialogue between the two agents using the set of dialogue acts $\mathcal{M}_{\mathcal{DC}}$, $\mathcal{M}'$ or $\mathcal{M}'_p$ will not terminate.*

Proof. Let us assume that P starts the dialogue. By definition P will *question* if possible, and since there are no dialogue conditions in force at the start of a dialogue will do so. The protocol allows C to respond with another question, and so, as an Elephant's Child, it will. By the same reasoning, P will then respond with a question, as will C, and so on. Even with a finite set of propositions available to them, each agent can generate an infinite number of questions, and the dialogue will never terminate. $\square$

Thus, as we might expect, agents which follow the Elephant's Child strategy can disrupt dialogues, but the following result seems to suggest that it takes two such agents to cause non-termination:

Theorem 15. *Given two agents P, which uses the Elephant's Child strategy, and C, any dialogue using the set of dialogue acts $\mathcal{M}_{\mathcal{DC}}$ can be made to terminate if C chooses appropriate acts.*

Proof. Consider that P moves first, as above issuing a *question*. For P to be able to issue another *question*, and so keep the dialogue running, C must issue a *question* or an *accept*. For this proof it is sufficient to consider ways in which C can prevent this happening. Initially, therefore C will respond to the question with an *assert* rather than a *question*. Considering the dialogue conditions on *assert*, and those on every legal act that may follow *assert*, there is no way that C can follow its assertion with an *accept* and thus no way that P can force itself into a position in which it can *question*. Similarly, if C starts the dialogue, provided it does not *question* or *challenge*, there is no sequence of moves P can make in response to C's initial move (which with $\mathcal{M}_{\mathcal{DC}}$ has to be an *assert*) which will mean that P is in a position to ask even one question. $\square$

It is currently an open question whether this latter result translates to $\mathcal{M}'$ and $\mathcal{M}'_p$.

Table 1. The knowledge bases for the example

	Σ_P	Σ_C
1	$\neg agr, pri, min$	$min, min \rightarrow \neg pri$
2	$pri \wedge \neg agr \rightarrow \neg pub$	pri
3	$min \rightarrow \neg pri$	

Table 2. The way the commitment stores change during the privacy dialogue

Move	CS
⟨P, C, $assert(\neg pub)$⟩	$CS.Kb_P = \{\neg pub\}$, $CS.Pref_P = \emptyset$ $CS.Kb_C = \emptyset$, $CS.Pref_C = \emptyset$
⟨C, P, $challenge(\neg pub)$⟩	$CS.Kb_P = \{\neg pub\}$, $CS.Kb_C = \emptyset$
⟨P, C, $assert(\{pri, \neg agr, pri \wedge \neg agr \rightarrow \neg pub\})$⟩	$CS.Kb_P = \{\neg pub, pri, \neg agr, pri \wedge \neg agr \rightarrow \neg pub\}$ $CS.Kb_C = \emptyset$
⟨C, P, $assert(\{min, min \rightarrow \neg pri\})$⟩	$CS.Kb_P = \{\neg pub, pri, \neg agr, pri \wedge \neg agr \rightarrow \neg pub\}$ $CS.Kb_C = \{min, min \rightarrow \neg pri\}$
⟨P, C, $prefer((pri, min \rightarrow \neg pri))$⟩	$CS.Pref_P = \{(pri, min \rightarrow \neg pri)\}$ $CS.Pref_C = \emptyset$
⟨C, P, $prefer((min \rightarrow \neg pri, pri))$⟩	$CS.Kb_P = \{\neg pub, \text{pri}, \neg agr, pri \wedge \neg agr \rightarrow \neg pub\}$ $CS.Pref_P = \{(pri, min \rightarrow \neg pri)\}$ $CS.Kb_C = \{min, min \rightarrow \neg pri\}$ $CS.Pref_C = \{(min \rightarrow \neg pri, min)\}$

4 An Example

In this section we present an example of a *persuasion dialogue* between two software agents P and C both of which use the instantiation of our general framework which was presented in the previous section. In this dialogue C is regarded as knowing more about the subject than P and so C $\succ$ P.

P: Newspapers can't publish the information X.
C: Why?
P: Because it's about A's private life and A doesn't agree.
C: But A is a minister, and any information concerning a minister is public.
P: I know, but that is less important than the private life of a person.
C: No, in politics the occupation is more important than anything else.

To handle this dialogue formally, we give the agents the knowledge-bases in Table 1 where *agr* denotes "A agrees", *pri* denotes "private", *min* denotes "A is a minister", and *pub* denotes "Newspapers can publish X").

This formal dialogue proceeds as in Table 2. The dialogue begins when P presents an argument, $A = (\{pri, \neg agr, pri \wedge \neg agr \rightarrow \neg pub\}, \neg pub)$, in favour of not publishing. That argument is initially acceptable in P's argumentation framework since it defends itself against the unique undercutting argument, $B = (\{min, min \to \neg pri\}, \neg pri)$. Later C gives its preferences and these conflict with P's. Since C is regarded as more reliable than P in this domain, P's argumentation framework will find that the argument A is not longer acceptable since in the most preferred context (that of agent C), B is preferred to A.

5 Conclusion

This paper makes two main contributions to the study of inter-agent dialogues. First, it presents a way of handling dialogues between agents with different preferences, which relaxes one of the more restrictive constraints imposed by much previous work in this area ([5] being an honourable exception). Second, it presents both a general framework which identifies and defines some of the important components in any agent dialogue, and a detailed description of an instantiation of this framework. This work is also novel, going further than previous attempts (including the work which underpins much of this paper [2,3]) in identifying all the parts of the computational framework necessary to generate dialogues between agents. We note that it leaves unresolved the way in which the illocutions we use relate to agent communication languages, and this something we aim to clarify in the future.

In some respects the work presented here follows on from the dialogical framework of Noriega and Sierra [13] and the framework for argumentation-based negotiation of Sierra *et al.* [17]. This paper starts at the same high level of abstraction, but goes into much greater detail about the precise way in which the agents carry out the argumentation which underpins the dialogue. This is not to say that the current work subsumes the ideas of dialogical frameworks and, especially, the related notion of electronic institutions [6]. Indeed, relating what we have here to the notion of institutions is the subject of ongoing work.

Acknowledgments

This work was funded by the EU SLIE Project (IST-1999-10948). Many thanks to Peter McBurney for helpful comments.

References

1. L. Amgoud, C. Cayrol, and D. LeBerre. Comparing arguments using preference orderings for argument-based reasoning. In Proceedings of the 8th International Conference on Tools with Artificial Intelligence, pages 400–403, 1996.
2. L. Amgoud, N. Maudet, and S. Parsons. Modelling dialogues using argumentation. In Proceedings of the International Conference on Multi-Agent Systems, pages 31–38, Boston, MA, 2000.
3. L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue, and negotiation. In Proceedings of the Fourteenth European Conference on Artificial Intelligence, pages 338–342, Berlin, Germany, 2000.

4. L. Amgoud, S. Parsons, and L. Perrussel. An argumentation framework based on contextual preferences. In Proceedings of the International Conference on Formal and Applied and Practical Reasoning, pages 59–67, 2000.
5. G. Brewka. Dynamic argument systems: a formal model of argumentation process based on situation calculus. Journal of Logic and Computation, 11(2):257–282, 2001.
6. M. Esteva, J. A. Rodríguez-Augilar, J. L. Arcos, C. Sierra, and P. Garcia. Institutionalising open multi-agent systems. In Proceedings of the 4th International Conference on Multi Agent Systems, pages 381–382, 2000.
7. T. F. Gordon. The pleadings game. Artificial Intelligence and Law, 2:239–292, 1993.
8. R. Kipling. Just So Stories. Everyman Library, 1992.
9. C. L.Hamblin. Fallacies. Methuen, London, 1970.
10. J. MacKenzie. Question-begging in non-cumulative systems. Journal of Philosophical Logic, 8:117–133, 1979.
11. P. McBurney and S.Parsons. Tenacious tortoises: A formalism for argument over rules of inference. In Worshop of Computational Dialectics: Models of argumentation, Negotiation, and Decision Making. 14th European Conference on Artificial Intelligence, 2000.
12. R. McConachy and I. Zukerman. Dialogue requirements for argumentation systems. In Proceedings of IJCAI'99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems, 1999.
13. P. Noriega and C. Sierra. Towards layered dialogical agents. In J. P. Muller, M. J. Wooldridge, and N. R. Jennings, editors, Intelligent Agents III, pages 173–188, Berlin, Germany, 1997. Springer Verlag.
14. S. Parsons, C. Sierra, and N. R. Jennings. Agents that reason and negotiate by arguing. Journal of Logic and Computation, 8(3):261—292, 1998.
15. S. Parsons, M. Wooldridge, and L. Amgoud. An analysis of formal inter-agent dialogues. Technical report, Department of Computer Science, University of Liverpool, 2001.
16. H. Prakken. On dialogue systems with speech acts, arguments, and counterarguments. In 7th European Workshop on Logic for Artificial Intelligence, Malaga, 2000.
17. C. Sierra, N. R. Jennings, P. Noriega, and S. Parsons. A framework for argumentation-based negotiation. In Proceedings of the 4th International Workshop on Agent Theories, Architectures and Languages, 1997.
18. K. Sycara. Persuasive argumentation in negotiation. Theory and Decision, 28:203–242, 1990.
19. F. Tohmé. Negotiation and defeasible reasons for choice. In Proceedings of the Stanford Spring Symposium on Qualitative Preferences in Deliberation and Practical Reasoning, pages 95–102, 1997.
20. S. Zabala, I. Lara, and H. Geffner. Beliefs, reasons and moves in a model for argumentation dialogues. In Proceedings of the Latino-American Conference on Computer Science, 1999.

An Abstract Machine for Classes of Communicating Agents Based on Deduction

Pierre Bonzon

HEC, University of Lausanne
1015 Lausanne, Switzerland
pierre.bonzon@hec.unil.ch

Abstract. We consider the problem of defining executable runs for classes of communicating agents. We first define an abstract machine that generates runs for individual agents with non-deterministic plans. We then introduce agent classes whose communication primitives are based on deduction. While current communication models are overly expressive with respect to the core agent models that are used as background theory, communicating agents based on deduction achieve a balanced integration. Contrary to other more theoretical work, their operational semantics are given by an abstract machine that is defined purely in sequential terms. This machine readily offers straightforward opportunities for implementing and experimenting prototypes of collaborative agents.

1 Introduction

According to the *theory of knowledge* [4], communication can be viewed as the act of upgrading the state of knowledge in a multi-agent system. At the low end of the spectrum is *distributed knowledge.* This situation arises when the deduction of some fact by a single agent requires information that is disseminated among the other agents. At the high end, *common knowledge* implies publicity, i.e. full reciprocal awareness of some fact by all agents. A system's performance obviously depends on its state of knowledge. Accordingly, in many applications, the focus is on trying to upgrade the system's state of knowledge through communication.Towards this end, an approach is to rely on an external model of knowledge based on possible world semantics. But in this solution there is "no notion of the agents computing their knowledge and no requirements that the agents be able to answer questions based on their knowledge"[4]. Agents should however be able to compute, and not just communicate their knowledge.

Turning away from the theoretical approach just outlined, practical agent communication models (such as those advocated by KQML and ACL of FIPA) are generally based on *speech act theory* [11]. They thus rely on the mental attitudes of agents. Ideally, these communication models should be coupled with comprehensive core agent models enjoying a minimum "understanding" for these various attitudes. Unfortunately, the current agent models that can be used as background theory are not so expressive. They generally lack many of the required "mind components" (e.g., those corresponding to such actions as making an offer, a promise, a request, etc.).

Most current multi-agent models therefore integrate poorly expressive core agent models with inadequate, overly expressive communication models. As a result, many proposed communicative actions are difficult (if not impossible) to match with the agent's semantics. While balanced integration should be sought, we do not know of any attempt to establish a formal correspondence between subsets of KQML or ACL, on one hand, and an agent model comprising at least *beliefs*, *desires* and *intentions*, such as *AgentSpeak(L)* [10]) or any other instance of the BDI model, on the other hand. Current literature on rational agents [12] does not even mention the problem.

Recently, a completely new approach has been advocated by Hindriks and al. [6]. Their logical communicative primitives do not correspond to any speech act in particular. They are defined as simple and "neutral" actions enjoying a well-defined and clear semantics that can be used for many different purposes, including the implementation of speech acts. Hindriks and al. further argue that speech act theory should not be viewed as a repository for all kinds of different communicative acts. Computational equivalents for speech acts do not necessarily have to be included in an agent communicative language, as done in KQML or ACL. Speech act theory may instead provide a set of abstract descriptions of communicative actions. These should be used then to specify and verify the behaviours of agents. It would then be up to the programmer to satisfy this specification using basic communicative actions.

An important feature in this new approach is the use of synchronised pairs: in order for two agents to communicate, both parties must first agree to an exchange (e.g., by independently using protocols based on these synchronised pairs). They will then wait until the exchange is completed before proceeding with their remaining activities. As an example (that will be developed later), this can be used in collaborative models to synchronise successive negotiation rounds as well as the successive steps involved in each round.

We shall follow and further simplify this logical approach. In short, Hindriks and al. consider the exchange of two messages between a *sender* and a *receiver* as a way for the receiver, either to use data provided by the sender to answer a query of his own, or to use his data to answer a query from the sender. These two types of messages are represented by two pairs i.e., *tell/ask* and *req/offer* respectively. In both cases, no data is sent back to the sender, and the formal semantics captures the processing done by the receiver (roles however may be switched, as we shall illustrate). This processing requires either a *deductive* (for *tell/ask*) or *abductive* (for *req/offer*) reasoning based on the receiver local state.

Deduction is a well understood task for which semi-decidable procedures can be easily implemented (e.g. under the form of a meta-interpreter within a logic programming framework). Abduction is a much more complex and difficult process. In order to define and implement *executable* runs involving deductions only, we shall give up the pair *req/offer* and define instead a simplified *call/return* pair that, similarly to *tell/ask*, relies on deduction only. When used in combination with the *tell/ask* pair, this new simplified pair will lack the full power of abduction. It still allows for the implementation of various communication protocols.

As logical communicative primitives do not involve mental attitudes, the corresponding core agent model can be kept simple. In the 3APL language of Hindriks and al., individual agents are multi-threaded entities consisting of *beliefs* and *goals*. This means that an agent may have multiple goals that are executed in parallel. A multi-agent system itself is again a multi-threaded system of multi-threaded entities. The corresponding operational semantics rely on concurrent programming concepts that are left implicit and thus achieve a seamless integration of the communication and the core agent models. But unless we first implement a true concurrent programming language that in addition offers all the required functionality (including deductive reasoning capabilities), we are left short of executable specifications.

We believe however that is both possible and worthwhile to try and get executable specifications by defining the complete model in purely sequential terms. The first issue we face is the choice of a core agent model. As already mentioned above, logical communication primitives do not require mental attitudes. One therefore does not need to distinguish between *goals* and *beliefs*.

In order to plan an agent's actions, at least two competing approaches are possible i.e., *static* planning and *reactive* (or dynamic) planning. While static planning involves explicit goals and means-end analysis, reactive planning is based on conditions-action rules without explicit goals. The logical specification of an agent's primitive actions required by static planning is an unrealistic prerequisite. Agents are not likely to reason about the effect of their actions on the environment. Furthermore, they will generally not react to environmental changes by designing complex plans from scratch. We therefore favour the reactive approach. In this framework, agents select rules from sets of *predefined* plans. As agents must be ready to reconsider their choices at any time, the issue is to enforce timely reactions leading to an appropriate *change* of plans.

Towards this end, we propose to incorporate the concept of plan within an existing model of reactive agents. Following Wooldridge & Lomuscio [13], a general model of agent with sensing can be given in abstract functional terms. In order to get concrete executable specifications, we shall first develop these functional definitions into a set of procedures. These procedures will represent an abstract machine generating runs for individual non-deterministic agents. The concept of plan is introduced next. Given by *logical implications* similar to conditions-action rules, an agent's set of predefined plans can be looked at as a logical agent's program. As in other logical agent models, the agent must first deduce the action it intends to take. But in contrast to other logical agent models e.g., such as Golog and/or ConGolog that are based on the situation calculus [2] [8], our agents deduce only one action at a time. We believe that this framework offers a valuable alternative to the approach just mentioned, especially if reactivity and communication are at stake. To substantiate this claim, we shall use our approach to implement our model of communicating agents based on deduction.

We shall not concern ourselves with the corresponding declarative semantics, and will be content with the operational semantics defined by the abstract machine. The benefits that follow from this approach are:

– extensions can be built on top of this machine: as an example, we will define and implement a model of multi-agent system as interleaving of individual agent runs
– the synchronisation operations for integrating this core system with the communication part can be made explicit and put under the agent's control
– using a step-wise refinement approach, this abstract machine may be implemented on any platform and in any particular programming environment: this is illustrated in the appendix outlining a Prolog implementation.

In summary, the extension of an agent model with sensing to include non-deterministic plans, the reduction of communication primitives to deductive reasoning, and their integration within a concrete multi-agent system are the main contributions of this paper.

The rest of this paper is organised as follows: in section 2, we reproduce the functional definition of an agent's run with sensing. Section 3 proposes a corresponding concrete model. Section 4 introduces agents with plans. Section 5 defines agent classes whose communication is based on deduction.

2 Abstract Functional Definitions

Following Wooldridge & Lomuscio [13], an environment *Env* is a tuple $\langle E, vis, \tau_e, e_0\rangle$, where

- $E=\{e_1, e_2, \ldots\}$ is a set of *states* for the environment
- $vis: E \rightarrow 2^E$ is a *visibility* function
- $\tau_e: E \times Act \rightarrow E$ is a *state transformer* function for the environment, with *Act* a set of *actions*
- $e_0 \in E$ is the *initial state* of the environment

and an agent *Ag* is a tuple $\langle L, Act, see, do, \tau_a, l_0\rangle$, where

- $L=\{l_1, l_2, \ldots\}$ is a set of *local states* for the agent
- $Act=\{a_1, a_2, \ldots\}$ is a set of *actions*
- $see: vis(E) \rightarrow Perc$ is the *perception* function mapping *visibility sets* to *percepts*,
- $\tau_a: L \times Perc \rightarrow L$ is the *state transformer* function for the agent
- $do: L \rightarrow Act$ is the *action selection* function, mapping agent local states to *actions*
- $l_0 \in L$ is the *initial state* for the agent.

An *agent system* is a pair *{Env,Ag}* whose set of *global states G* is any subset of $E \times L$. A *run* of a agent system is a (possibly infinite) sequence of global states $(g_0, g_1, g_2, \ldots)$ over G such that

- $g_0 = \langle e_0, l'_0\rangle$, with $l'_0 = \tau_a(l_0, see(vis(e_0)))$
- $\forall u$, if $g_u = \langle e_u, l'_u\rangle$, then $g_{u+1} = \langle e_{u+1}, l'_{u+1}\rangle$, with $e_{u+1} = \tau_e(e_u, do(l'_u))$ and $l'_{u+1} = \tau_a(l'_u, see(vis(e_{u+1})))$.

3 A Concrete Model of Non-deterministic Agents with Sensing

Let S be the set of sentences of first order logic with arithmetic whose set of predicates includes the predicate *do/1*, and let $L = 2^S$ and $Perc=S$. If we incorporate the selection of actions and the mapping of visibility sets within the functions τ_e and τ_a, then we get two new functions $\tau_{e,do} : E \times L \rightarrow E$ and $\tau_{a,see,vis} : L \times E \rightarrow L$. Equivalently, these new functions can be seen as procedures with side effects i.e.,

$\tau_{e,do} : E \times L \rightarrow E \Rightarrow$ ***procedure*** *react(e,l)* with side effects on *e*

$\tau_{a,see,vis} : L \times E \rightarrow L \Rightarrow$ ***procedure*** *sense(l,e)* with side effects on *l*.

We can define these procedures as follows

```
procedure react(e,l )
if  l ⊢ do(a)
then  e ← $\tau_e(e,a)$
```

```
procedure sense(l,e)
if "the environment produces percept s"
then  l ← $\tau_a(l,s)$
```

We write $l \vdash do(a)$ to mean that the formula *do(a)* can be proved from the formula *l*, meaning in turn that *a* is an applicable action. An agent's run is defined by

```
procedure run(e,l)
loop  sense(l,e);
      react(e,l)
```

Although environments are supposed to evolve deterministically, the choice to be made among possible actions, i.e. among all a_i such that $l \vdash do(a_i)$, is left unspecified. Consequently, the *run* procedure can be seen as a *non-deterministic* abstract machine generating runs for logical agents. In short, it is a concrete model of non-deterministic agents.

4 Non-deterministic Agents with Plans

Intuitively, an agent's plan can be described as an ordered set of actions that may be taken, in a given state, in order to meet a certain objective. As above, agents whose choice among possible plans (i.e. those that are applicable in a given state) is left unspecified are non-deterministic. Let us assume that the set of constant symbols and predicates of *S* include a set $P = \{p_1, p_2, \ldots\}$ of non-deterministic plan names (*nd-plan* in short) and three predicates *plan/1, do/2* and *switch/2*. Let us further extend the definition of an agent's global state to include its current active nd-plan *p*. We finally have the following new procedures:

```
procedure react(e,l,p)
if  l ⊢ do(p, a)
then e ← $\tau_e(e,a)$
else if l ⊢ switch(p, p′)
     then react(e,l,p′)
```

```
procedure run(e,l )
loop  sense(l,e);
      if l ⊢ plan($p_0$)
      then react(e,l,$p_0$)
```

In each *react* call, the agent's first priority is to carry out an action *a* from its current plan *p*. Otherwise, it may switch from *p* to *p′*. When adopting a new plan, a recursive call to *react* leads in turn to the same options. In *run,* the initial plan p_0 is chosen in each cycle. If the environment has not changed in between, then the agent is bound to adopt its last active plan again. On the other hand, if the environment has changed, and if the embedding of plans reflects a hierarchy of priorities (i.e., the structure that can be associated with the *switch/2* predicate is that of directed *acyclic* graphs rooted at each initial plan) then it will select a plan that has the *highest implicit priority*.

This achieves a simple way to instruct an agent to adopt a new plan whenever a certain condition occurs, without having to tell it explicitly when to switch from its current plan. The corresponding switching logic is thus easier to define than if the last active plan was kept at each cycle: in this case, the *run* procedure would involve less overhead, but explicit switching conditions should be given for each plan an agent may switch to from its current plan (i.e., the structure associated with the *switch/2* predicate would be that of directed *cyclic* graphs).

Example

Let us consider the mail delivery robot of LespØrance & al [7]. Let *start* be its single initial plan. Its first priority is to handle a new order. It will then unconditionally switch to plan *check* to see if any order has to be cancelled. Depending on his current state, it will then switch to either plan *move* (and go on moving without conditions) or to plan *motionControl* (and search for a new customer). This second plan will lead in turn to switch to either *tryServe* or *tryToWrapUp*. Should a new order or a cancellation arise while it is attending other business, it will then automatically switch to the corresponding plan even though there are no explicit switching conditions for doing so. This set of plans is given by the following implications and facts:

orderState(N,justIn) ⇒ *do(start,handleNewOrder(N))*

switch(start,check)

orderState(N,toPickUp) ∧sender(N,Sender) ∧ suspended(Sender)
⇒ *do(check,cancelOrder(N))*

robotState(moving) ⇒ *switch(check,move)*

do(move,noOp(moving))

¬ robotState(moving) ∧ (orderState(N,toPickUp) ∨ orderState(N,onBoard) ∨
¬ robotPlace(centralOffice)) ⇒ *switch(check,motionControl)*

¬ searchedCustomer ⇒ *do(motionControl,searchCustomer),*

customerToServe(Customer) ⇒ *switch(motionControl,tryServe(Customer)),*

¬ customerToServe(_) ⇒ *switch(motionControl,tryToWrapUp),*

robotState(idle) ⇒ *do(tryServe(Customer),startGoto(mailbox(Customer)))*

robotState(stuck) ⇒ *do(tryServe(Customer),resetRobot)*

robotState(reached) ⇒ *do(tryServe(Customer),freezeRobot)*

robotState(frozen) ⇒ *do(tryServe(Customer),dropOffShipmentsTo(Customer))∧*
do(tryServe(Customer),pickUpShipmentsFrom(Customer)) ∧
do(tryServe(Customer), resetRobot)

robotState(idle) ⇒ *do(tryToWrapUp,startGoto(centralOffice))*

etc…

A Particular Case: Priority Processes

When nd-plans form equivalence classes that can be linearly ordered, these classes can be identified with plans of equal *priority.* If priorities are represented by positive integers *n*, then we get a new reaction scheme without explicit switching conditions, where plans define *priority processes 1,2,..,n* . This leads the new procedure

```
procedure process(e,l,n)
if  $l \vdash do(n, a)$
then $e \leftarrow \tau_e(e,a)$
else if $n > 0$
    then process(e,l,n-1)
```

If procedure *process* is called repeatedly with the highest priority, then the execution of a process *n* will proceed unless the conditions for a process with a higher priority become satisfied. Since we have $l \vdash do(n, a)$, it can be assumed that once a process *n* is selected, then at least one of its action will be executed. Priority processes and plans can be interleaved in many ways. As an example let us consider the following run^+ procedure relying on a new predicate *priority/1* delivering the *current highest priority* n_0:

```
procedure $run^+(e,l)$
loop    sense(l,e);
        if $l \vdash priority(n_0)$
        then $process(e,l,n_0)$;
        if $l \vdash plan(p_0)$
        then $react(e,l,p_0)$
```

and let use it to implement the mail delivery robot in a way that is very similar to the solution given by LespØrance& al. using ConGolog [7]. The top plans (up to but not including plans *tryServe* and *tryToWrapUp*) can be expressed as priority processes:

$orderState(N,justIn) \Rightarrow do(3,handleNewOrder(N))$

$orderState(N,toPickUp) \wedge sender(N,Sender) \wedge suspended(Sender)$
$\Rightarrow do(2,cancelOrder(N))$

$\neg robotState(moving) \wedge (orderState(N,toPickUp) \vee orderState(N,onBoard) \vee$
$\neg robotPlace(centralOffice)) \Rightarrow do(2,robotMotionControl)$

$robotState(moving) \Rightarrow do(1,noOp)$

In order to activate one the remaining plans, the *robotMotionControl* action must allow for the assertion, within *l,* of either *plan(tryServe(Customer))* or *plan(tryToWrapUp).* A comparison with the ConGolog solution reveals that:

- in our solution, the entire control mechanism is given in terms of nd-plans and/or processes, whose execution steps are *explicitly* interleaved with sensing: as a result, autonomous agents whose independent, asynchronous actions must be co-ordinated and/or synchronised may be easily implemented
- in the ConGolog solution, the *mainControl* procedure concurrently executes four ConGolog interrupts that corresponds to our four *priority processes,* but robot motion control relies on sequential procedures that run asynchronously with the rest of the architecture; the environment is thus *implicitly* monitored.

5 Communicating Agents Based on Deduction

As an example of autonomous agents whose independent actions must be synchronised, let us now define and implement a model of communicating agents. As indicated in the introduction, we shall use and simplify the proposal made by Hindriks and al. [6]. Following a purely logical approach, they introduce two pairs of neutral communication primitives i.e., *tell/ask* and *req/offer*, that correspond to data exchanges enjoying a well-defined semantics and can be used for many different purposes.

In each pair, r is designated as the *receiver* and s as the *sender*. In the first exchange, message *tell*(r,φ) from sender s provides r with data φ, and message *ask*(s,ψ) from receiver r expresses his willingness to solve his own query ψ using any data sent by s. Both messages are sent asynchronously, without reciprocal knowledge of what the other agent wants or does. In particular, the data φ volunteered by s is not given in response to rs asking. If these two messages are put together through some kind of a handshake or synchronisation, then by using both his own knowledge and the data φ told by s, receiver r will try and answer his query ψ. Formally, receiver r will *deductively* compute the most general substitution θ such that

$$l^r \cup \varphi \vdash \psi\theta \; .$$

According to Hindriks & al., ψ in ask(s,ψ) can contain free variables but φ in *tell*(r,φ) must be closed; furthermore, $l^s \vdash \varphi$ is not required (i.e., s is not required to be truthful or honest). We shall illustrate this type of exchange through a simple example. Let the local state l^r of r be such that

$$l^r \vdash father(abram,isaac) \wedge father(isaac,jacob)$$

and let us consider the following pair of messages

message sent by s: $tell(r, \forall XYZ\, father(X,Y) \wedge father(Y,Z) \Rightarrow grandfather(X,Z))$
message sent by r: $ask(s, grandfather(X,jacob))$.

In this first scenario, s tells r a closed implication, and r asks s for some data that could allow him to find out who is the grandfather of *jacob*. Using the data sent by s, r is then able to deduce the substitution $X{=}abram$.

In contrast, message *req*(r,φ) from sender s requests r to solve query φ, and message *offer*(s,ψ) from receiver r expresses his willingness to use his own data ψ for solving any query submitted by s. When put together, these two messages will allow the receiver r to find the possible instantiations of his free variables in ψ that allow him to deduce φ. Formally, receiver r will *abductively* compute the most general substitution θ such that

$$l^r \cup \psi\theta \vdash \varphi \, .$$

According to Hindriks & al., φ in *req*(r,φ) must be closed but ψ in *offer*(s,ψ) can contain free variables; furthermore, $l^r \vdash \psi$ is not required, but $l^r \vdash \neg\psi$ is not allowed. To illustrate this second type of exchange, let us consider the following pair of messages

message sent by s: $req(r, \exists X\, grandfather(X,jacob))$
message sent by r: $offer(s, father(X,Y) \wedge father(Y,Z) \Rightarrow grandfather(X,Z))$.

In this second scenario, *s* requests *r* to find out if there is a known grandfather for *jacob.* Independently, *r* offers *s* to abduce a substitution for the free variables in his ψ that would allow him to answer. In this case, using his knowledge contained in l^r and the implication he offers, *r* can abduce the same substitution as before.

In both of the above exchanges, no data is sent back to *s*, and the corresponding formal semantics captures the processing done by *r* only. In other words, the sender will not be aware of the results of the receiver's computation. For the sender to get this results, a subsequent reversed exchange (e.g. *ask/tell*) is needed. While this is perfectly appropriate for the first type of exchange (after-all, the sender who volunteers data is not necessarily interested the receiver's computations), we feel that the sender who, in the second case, expresses a need for data should automatically benefit from the receiver's computations. Furthermore, as abductions are difficult to achieve and implement, we favour exchanges that do not rely on abduction. Giving up the *req/offer* pair, we shall thus define and implement instead a simplified *call/return* pair that relies on deduction only. By doing so, we will end up with a less powerful model. It is interesting to note however that all *req/offer* examples given b Hindriks & al. 99 can be expressed as *call/return* invocations. In particular, if the receiver's local state includes closed forms of his offer ψ, then a *req*(r,φ)/*offer*(s,ψ) pair reduces to a *call/return* pair (this will also be illustrated at the end of this section).

In the new *call*(r,φ)/*return* (s,ψ) pair, both φ and ψ can contain free variables. This exchange is then interpreted as the sender *s* calling on *r* to instantiate the free variable in his query φ. Independently, the receiver *r* is willing to match his query ψ with the sender's φ and to return the instantiations that hold in his own local state. Formally, receiver *r* will *deductively* compute the substitutions θ such that

$\varphi\theta = \psi\theta$ and $l^r \vdash \psi\theta$.

To illustrate this, let us suppose that we now have

$l^r \vdash father(abram,isaac) \wedge father(isaac,jacob) \wedge$
$\quad \forall XYZ\, father(X,Y) \wedge father(Y,Z) \Rightarrow grandfather(X,Z)$

message sent by *s*: *call(r, grandfather(X,jacob))*
message sent by *r*: *return(s, grandfather(X,Y)).*

This exchange is to be interpreted as *s* calling on *r* to find out the grandfather of *jacob* i.e., to instantiate the free variable in his query. Independently, *r* is willing to match the sender's call and to return the substitutions that hold in his local state. Once again the substitution *X=abram* will be found. In contrast to the previous exchanges however, this information will be sent back to the sender.

A Concrete Model and Its Implementation

The core model we shall adopt consists of *classes* of identical agents, as defined in section 4. Similarly to classical *object* theory we shall distinguish the *class itself*, considered as an object of type "*agent class*", and its class *members* i.e., the objects of type "*agent instance*". The class itself will be used both as a *repository* for the common properties of its members and as a *blackboard* for agent communication.

Messages exchanged between class members must use a *data transport* system. We shall abstract this transport system as follows: any message sent by an agent (this message being necessarily half of an exchange as defined above) will be first posted in the class. The class itself will then interpret the message s contents, wait for the second half of the exchange (thus achieving synchronisation), and finally perform the computation on behalf of the receiver. As an assumption, each message will be *blocking* until the exchange s completion i.e., no other exchange of the same type will be allowed between the sender and the receiver before the exchange is completed.

In order to further simplify our presentation, we shall consider purely communicating agents i.e., agents that do not carry out any action other than the exchange of messages. The environment per se will thus be ignored. Formally, the *local state* of a class of agents seen as a whole i.e., including its members, will be defined by a vector $l = [l^{Class}, l^1 \ldots l^n]$, where the components l^{Class} and l^i are the local state of the class itself and of its members identified by an integer $i=1 \ldots n$, respectively. We will use a new predicate *agent/1* and assume that $l^{Class} \vdash agent(i)$ whenever agent *i* belongs to the class.

As communicative actions do not affect the environment, the state transformer function $\tau_e: E \times Act \rightarrow E$ should be replaced by a function $\tau_a: L \times Act \rightarrow L$. Actually, in order to take into account the originator of a communicative action, we shall consider a set of such transformer functions, each function being associated with a given class *Class* or member *i*. We will thus consider the function $\tau^{Class}: L \times Act^{Class} \rightarrow L$ to be used in procedure $process^{Class}$, on one hand, and the functions $\tau^i : L \times Act^i \rightarrow L$, $i=1,\ldots,n$, to be used in procedure $react^i$, on the other.

The abstract machine that defines the run of a class of agents as *interleavings* of individual runs is then defined by the following procedure:

```
procedure $run^{Class}(l)$
loop  for all  i such that  $l^{Class} \vdash agent(i)$ do
            if $l^i \vdash plan(p_0^i)$
            then $react^i(l, p_0^i)$;
        if $l^{Class} \vdash priority(n_0)$
        then $process^{Class}(l, n_0)$
```

In this particular definition, messages are first processed (via the calls to $react^i$) and then possibly synchronised without delay (via the subsequent call to $process^{Class}$). In this framework, the $react^i$ and $process^{Class}$ procedures are defined as follows:

```
procedure $react^i(l, p^i)$
if   $l^i \vdash do(p^i, a)$
then  $l \leftarrow \tau^i(l, a)$
else  if $l^i \vdash switch(p^i, p^{i\prime})$
      then $react^i(l, p^{i\prime})$

procedure $process^{Class}(l, n)$
if   $l^{Class} \vdash do(n, a)$
then  $l \leftarrow \tau^{Class}(l, a)$
else  if $n > 0$
      then $process^{Class}(l, n-1)$
```

The state transformer function τ^s required to process the message *tell(r,φ)* is:

$$\begin{array}{lll} \tau^s([l^{Class},\ldots l^s,\ldots],\ tell(r,\varphi)) & = & \textbf{if}\ busy(tell(r,\varphi)) \notin l^s \\ & & \textbf{then}\ [l^{Class} \cup \{ack(s,tell(r,\varphi))\},\ldots \\ & & \quad l^s \cup \{busy(tell(r,\varphi))\},\ldots] \\ & & \textbf{else}\ \ [l^{Class},\ldots\ l^s,\ldots] \end{array}$$

The functions for messages *ask(s,ψ)*, *call(r,φ)* and *return(s,ψ)* are similarly defined. Each message is thus first acknowledged by the class, a blocking flag (i.e., *busy*) is raised, and the message waits to be synchronised. Synchronisation occurs when two messages belonging to the same pair have been acknowledged. This synchronisation is triggered by two *priority processes* defined as:

$$\begin{array}{lll} ack(s,tell(r,\varphi)) \wedge ack(r,ask(s,\psi))) & \Rightarrow & do(2,\ tellAsk(s,r,\varphi,\psi)) \\ ack(s,call(r,\varphi)) \wedge ack(r,return(s,\psi)) & \Rightarrow & do(1,\ callReturn(s,r,\varphi,\psi)) \end{array}$$

The state transformer function τ^{Class} achieving synchronisation by the class is:

$$\begin{array}{l} \tau^{Class}([l^{Class},\ldots l^s,\ldots l^r,\ldots],tellAsk(s,r,\varphi,\psi))\ = \\ \qquad \textbf{if}\ l^r \cup \varphi \vdash \psi\theta \\ \qquad \textbf{then}\ \ [l^{Class} - \{ack(s,tell(r,\varphi)),\ ack(r,ask(s,\psi))\},\ldots \\ \qquad\qquad l^s\text{-}\{busy(tell(r,\varphi)),\ sync(\_)\} \cup \{sync(tell(r,\varphi))\},\ldots \\ \qquad\qquad l^r\text{-}\{busy(ask(s,\psi)),\ sync(\_)\} \cup \{sync(ask(s,\psi\theta\ ))\},\ldots] \\ \qquad \textbf{else}\ \ [l^{Class},\ldots\ l^s,\ldots l^r,\ldots] \end{array}$$

$$\begin{array}{l} \tau^{Class}([l^{Class},\ldots l^s,\ldots l^r,\ldots],callReturn(s,r,\varphi,\psi)) = \\ \qquad \textbf{if}\ \ \varphi\theta = \psi\theta\ \ \textbf{and}\ \ l^r \vdash \psi\theta \\ \qquad \textbf{then}\ [l^{Class} - \{ack(s,call(r,\varphi)),\ ack(r,return(s,\psi))\},\ldots \\ \qquad\qquad l^s\text{-}\{busy(call(r,\varphi)),\ sync(\_)\} \cup \{sync(call(r,\varphi\theta\ ))\},\ldots \\ \qquad\qquad l^r\text{-}\{busy(return(s,\psi)),sync(\_)\} \cup \{sync(return(s,\psi\theta\ ))\},\ldots] \\ \qquad \textbf{else}\ [l^{Class},\ldots\ l^s,\ldots l^r,\ldots] \end{array}$$

Old flags are removed and a new *sync* flag carrying the computation result is raised. To ensure simple sequential execution, a single such *synchronisation* flag is available at any time for each agent. Thus there will be no trace of successive exchanges, and agent s nd-plans must be designed to use this flag accordingly.

Example: Two-Agent Meeting Scheduling

In this simplified version of the *two-agent scheduling* example of Hindriks and al. [6], one agent is designated as the *host* and the other one as the *invitee*. Both agents have free time slots to meet e.g.,

$$\begin{array}{ll} l^{host} & \vdash\ meet(13) \wedge meet(15) \wedge meet(17) \wedge meet(18) \\ l^{invitee} & \vdash\ meet(14) \wedge meet(16) \wedge meet(17) \wedge meet(18) \end{array}$$

and they must find their earliest common free slot (in this case, *17*). The host has the responsibility of starting each *round* of negotiation with a given lower time bound *T*. A round of negotiation comprises three *steps*, each step involving in turn an exchange of messages.

In the first step (corresponding to the first line of both *invite* and *reply*), the host initialises a *call/return* exchange, calling on the invitee to find out his earliest free spot *T1* after *T*. In the second step (corresponding to the second line), the roles are swapped: the invitee initialises a *call/return* calling on the host to find out his earliest free spot *T2* after *T1*. In the final step the host either confirms an agreement on time *T2* (if *T1=T2*) by initialising a *tell/ask* exchange, or starts a new round with *T2*.

The corresponding host and invitee plans i.e., *invite(Invitee,T)* and *reply(Host),* are

sync(dialog(invite(Invitee,T)))
$\Rightarrow$ *do(invite(Invitee,T),call(Invitee,epmeet(T1,T)))*
sync(call(Invitee,epmeet(T1,T)))
$\Rightarrow$ *do(invite(Invitee,T),return(Invitee,epmeet(T2,T1)))*
sync(return(Invitee,epmeet(T2,T1)))$\wedge$*T=T2*
$\Rightarrow$ *do(invite(Invitee,T),tell(Invitee,confirm(T2)))*
sync(return(Invitee,epmeet(T2,T1)))$\wedge\neg$*(T=T2)*
$\Rightarrow$ *do(invite(Invitee,T),resume(invite(Invitee,T2)))*

sync(dialog(reply(Host)))
$\Rightarrow$ *do(reply(Host),return(Host,epmeet(T1,T)))*
sync(return(Host,epmeet(T1,T)))
$\Rightarrow$ *do(reply(Host),call(Host,epmeet(T2,T1)))*
sync(call(Host,epmeet(T2,T1)))$\wedge$ *T=T2*
$\Rightarrow$ *do(reply(Host),ask(Host,confirm(T2)))*
sync(call(Host,epmeet(T2,T1)))$\wedge\neg$*(T=T2)*
$\Rightarrow$ *do(reply(Host),resume(reply(Host)))*

where the flags *sync(dialog(invite(Invitee,T)))* and *sync(dialog(reply(Host)))* are used to initialise the exchange. Message *resume* (used by an agent to restart a plan) and predicate *epmeet(T1,T)* meaning "*T1 is the earliest possible meeting time after T*" are defined as

$$\tau^i([l^{Class},\ldots l^i,\ldots],\ resume(p)) \;=\; [l^{Class},\ \ldots l^i - \{sync(\_),plan(\_)\}\cup\{plan(p),sync(dialog(p))\},\ldots].$$

$$meet(T1)\wedge(T1>=T)\wedge\neg(meet(T0)\wedge(T0>=T)\wedge(T0<T1)) \Rightarrow epmeet(T1,T).$$

In comparison, Hindriks and al. alternate exchanges *req(Invitee, ∃ T1 epmeet(T1,T)) / offer(Host,meet(T2))* and *offer(Invitee,meet(T4)) / req(Host, ∃ T3 epmeet(T3,T2))* that involve abductive tasks. As ground instances (i.e., *meet(13), meet(14),* etc.) of the receivers offers are available in their respective local states, we can alternate instead *call/return* and *return/call* exchanges leading to the same result through simpler deductive tasks. As discussed in the introduction, 3APL synchronisation operations are left implicit. Therefore, the two 3APL "practical reasoning rules" that correspond to our *invite(Invitee,T)* and *reply(Host)* plans do not require any flag. They are however not directly executable by a sequential machine, whereas nd-plans are (see the appendix for the outline of a Prolog implementation). In our further work [1], plans are rewritten as *dialogs* with an implicit synchronisation. As these dialogs can be compiled back into nd-plans, they also represent executable specifications.

6 Related Work

A popular choice to specify executable agent models is to rely on the *situation calculus*. This extension of the first order calculus was designed to allow reasoning about the effect of actions. In the *Golog* system [7], which is based on this approach, an interpreter verifies if predefined programs are applicable to a given goal. If successful, it then delivers the execution trace corresponding to a sequence of actions that will fulfil this goal. The result is a situation that is considered a final state in a system of state transitions. *ConGolog* [2] represents a development of *Golog* in the direction of concurrent agent programming. In order to allow plans "to be suspended or terminated and new plans devised to deal with exceptional event or condition"[2], this extension introduces concurrent processes with priorities as well as interrupts. If an interrupt gets control from a higher priority process, this interrupt may trigger and its body is then executed (possibly repeatedly i.e., as long as its guard is satisfied). This computational model corresponds in many ways to our runs based either on nd-plans or priority processes. In particular, a *ConGolog* interrupt $<\varphi \rightarrow \sigma >$ associated with a process of priority *n* could be represented in our framework by an implication $\varphi \Rightarrow do(n,\sigma)$. However as ConGolog processes need not be linearly ordered, it may not be obvious how to assign them an explicit priority *n.* Since the basic computational mechanism of Golog/ConGolog is embedded in logic, it is possible to use a model of the action theory to assign semantics to programs. Being essentially at a meta-level, our approach does not allow this.

ConGolog original specifications as an offline interpreter did do not provide facilities for either agent sensing or agent communications, whereas our approach does. Our own framework being an online interpreter based on a reactive agent model, the comparison may thus be misleading in this respect. Recent proposals [3] to allow agents with sensing seem however to close the gap between these two approaches, as the on-line execution model of [3] no longer requires searching for a final state.

We have already sketched in the introduction alternative ways to model communication. We also indicated why we turned to the approach of Hindriks and al. We refer to them for a thorough discussion of the relationship between their proposal and current agent communication models.

7 Conclusion and Future Work

Most current communication models are overly expressive with respect to the available agent models that can be used as background theory. As a result, many proposed communicative actions are difficult (if not impossible) to match with a given agent's core semantics. Communicating agents based on simple deductive and/or abductive exchanges, as introduced by Hindriks and al. [6], achieve one of a few existing *balanced integration* we know of. These exchanges were further simplified in section 5 in order to rely on deduction only, and will apply in cases where the receiver's local state includes closed forms of his offers. Contrary to other more theoretical work, based for example on the π-calculus [5],[9], their operational semantics were given here by an abstract machine that is defined purely in sequential terms. It thus readily offers straightforward opportunities for implementing prototypes of collaborative agents. As an example, we have run a solution of the *n-agent* meeting problem (also discussed in [6]). This solution is given under the form of *dialogs* that can be *simply sequentially* executed by updating the single current synchronisation flag for each agent at each step. In order to achieve this result, dialogs expressed in a higher level language with implicit synchronisation must be first compiled into nd-plans.

At the same time, this framework will accommodate the extensions and/or refinements needed to develop full-fledged agents interfaced with real communication software. Towards this end, we are considering transporting the entire system experimented so far within standalone Prolog onto a commercial platform with built-in communication sockets and a Prolog logic server available as JAVA classes.

Acknowledgement

The author wishes to thank the anonymous referees for their valuable comments. Indeed most of these comments found their way into this revised version.

References

1. P. Bonzon, Compiling Agent Dialogs for Simple Sequential Execution, submitted
2. G. de Giacomo, Y.Lespérance and H. Levesque, ConGolog, a Concurrent Programming Language Based on the Situation Calculus, *Artificial Intelligence*, vol. 121 (2000)
3. G. de Giacomo and H. Levesque, An Incremental Interpreter for High-Level Programs with Sensing, in: H. Levesque & F. Pirri (eds), *Logical Foundations for Cognitive Agents*, Springer (2000)
4. R. Fagin, J. Halpern, Y Moses & M. Vardi, *Reasoning About Knowledge*, MIT Press (1995)
5. Ferber & O. Gutknecht, Operational Semantics of Multi-agent Organizations, in: N.R. Jennings and Y. Lespérance (eds*), Intelligent Agents VI*, LNAI, vol. 1757 , Springer (2000)
6. K.V. Hindriks, F.S. de Boer, W.van der Hoek and J.-J. Meyer, Semantics of Communicating Agents Based on Deduction and Abduction, *Proceedings IJCAI99 Workshop on ACL* (1999)
7. Y. Lespérance, K. Tam and M. Jenkin, Reactivity in a Logic-Based Robot Programming Framework, in: N.R. Jennings and Y. Lespérance (eds*), Intelligent Agents VI*, LNAI, vol. 1757, Springer (2000)
8. H.J.Levesque, R.Reiter, Y.Lespérance, F.Lin & R.Scherl, GOLOG: A Logic Programming Language for Dynamic Domains, *Journal of Logic Programming*, vol. 31 (1997)
9. R. Milner*, Communicating and Mobile Systems*; the π-Calculus, Cambridge Univ. Press (1999)
10. A.S. Rao, AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language, in: W. Van der Velde and J.W. Perram (eds.), *Agents Breaking Away* (MAAMAW '96), LNAI vol. 1038, Springer (1996)
11. R. Searle, *Speech Acts*, Cambridge University Press (1969)
12. M. Wooldridge, *Reasoning about Rational Agents*, MIT Press (2000)
13. Reasoning about Visibility, Perception and Knowledge, in: N.R. Jennings and Y. Lespérance (eds*), Intelligent Agents VI*, LNAI ,vol. 1757, Springer (2000)

Appendix: Towards a Prolog Implementation

Both an agent class and its members are represented by *objects* identified as *Class* and *Class(I),* respectively. The formulas *P* contained in an object are asserted as unit clauses *instance(Object,P).* Operations on objects are defined by the primitive procedures

```
new(Object)              :- retractall(instance(Object,_)).
insert(Object,P)         :- assert(instance(Object,P)).
remove(Object,P)         :- retractall(instance(Object,P)).
insertList(Object,L)     :- forall((L:List,member(P,List)),
                                   insert(Object,P)).
```

As individual agents inherit the properties of their class, each agent's local *state* encompasses both the *private* formulas that are contained in the agent itself, as well as the *public* formulas that are contained in its class. The *state(Object,P)* predicate meaning "*formula P is contained in the local state of Object*" is then defined as

```
state(Object,P)     :- private(Object,P);
                       public(Object,P).
private(Object,P)   :- instance(Object,P).
public(Object,P)    :- Object=Class(I),
                       instance(Class,P).
```

A meta-interpreter *ist(Object,P)* for simple deductions implementing a restricted form of $Object \vdash P$ is defined as

```
ist(Object,P)       :- state(Object,P).
ist(Object,Q)       :- state(Object,P=>Q),
                       ist(Object,P).
ist(Object,(P,Q))   :- ist(Object,P),
                       ist(Object,Q).
```

Methods representing class or agent actions are contained in the agent's class. Methods are terms *method(Object.Call,Body),* where *Call* is the name of a method followed by its parameters within parentheses and *Body* contains primitive procedure calls and/or *messages*. Messages sent to an *Object* have again the form *Object.Call,* where *Object* is either *Class* or *Class(I).* Messages are interpreted by

```
Object.Call      :-state(Object, method(Object.Call,Body)),
                   call(Body).
```

where *Call(Body)* represents a call to Prolog itself. According to this implementation, class actions are activated by messages sent to the class itself with *Object=Class* (e.g., in order to achieve synchronisation between agents), and agent actions are activated by messages sent to individual agents with *Object=Class(I)* .

Agent classes and class members are created with predefined messages inserting the required methods and the initial state into the corresponding objects. All procedures defining the sequential abstract machine are implemented as class methods. The run of an individual agent is then obtained by sending the message *Class(I).run*.

A Formal Semantics for ProxyCommunicative Acts

Marcus J. Huber[1], Sanjeev Kumar[1], Philip R. Cohen[1], and David R. McGee[2]

[1] Computer Science and Engineering, Oregon Graduate Institute
20000 NW Walker Road, Beaverton, OR 97006, USA
{marcush,skumar,pcohen}@cse.ogi.edu
[2] Battelle, Pacific Northwest National Laboratory, Richland, WA 99352
david.mcgee@pnl.gov

Abstract. Mediation services are becoming increasingly important in multiagent systems. An agent that can act on behalf of another agent is one important example of mediation functionality commonly required. Within this paper, we define and analyze PROXY and PROXY-WEAK communicative acts that formally specify semantics for interacting with middle agents that provide proxy services. These two communicative acts are shown to have a distinctly different impact upon the mental state of the agents involved and impose significantly different levels of commitment upon the middle agents.

1 Introduction

A common design attribute in many general-purpose multiagent software architectures and distributed computing environments is agents or processes whose sole purpose is to help locate other agents to find and communicate with them. These mediators or middle agents include the "facilitator" agents in the FIPA [5] and OAA [8] architectures, the "proxy" agents of the DARPA CoABS Project's Grid, and the proxy web servers on a network. Within KQML [4], it is common for an agent to use agents to "recruit" other agents that can provide it services.

These software proxies are an increasingly important aspect of distributed computational systems and are being introduced in a wide range of domains. In many cases, the notion entails a computational process that acts on behalf of, and typically assumes full responsibility for, the activities of another computational process (which ostensibly has some limitation that requires the use of the proxy). In other cases, the proxying entity should have no real responsibility. Unlike KQML's forward [4], proxying does not always entail simply passing on messages between entities that might not otherwise have contact with each other.

As an example of the difference in commitment levels that we are trying to support, recall the Watergate affair. President Nixon wanted the special prosecutor Archibald Cox to be fired, and asked Elliott Richardson (the Attorney General) to do so. Nixon wanted Richardson to take responsibility for the firing rather than just have Richardson tell Cox that Nixon is firing Cox.[1]

[1] As a point of interest, Richardson resigned to avoid being Nixon's proxy.

While we spend the beginning of the paper on details of the notation that we employ and the definition of key concepts, the focus of the paper is devoted to the definition and analysis of two new communicative acts, PROXY and PROXY-WEAK. Their employment should facilitate the deployment of mediation agents that provide proxying services. Our analysis below will show that the two acts result in the middle agents having significantly different levels of commitments relative to the final agents, where PROXY imposes significant and PROXY-WEAK imposes relatively little responsibility upon the middle agents. Our definition of PROXY-WEAK facilitates *third-party performative* semantics; we show that satisfaction and successful discharge of the PROXY-WEAK speech act is semantically equivalent to the sending agent performing a speech act directly on the final target agent even while going through an intermediary. As part of our analysis, we show that FIPA's two speech acts of the same names suffer from a number of deficiencies, including misrepresentation of the sender's true intentions and failure to provide third-party performative semantic support.

2 Background Concepts

We adopt an attempt-based semantics [2][3][12] for communication performatives in the following definitions. We refer to this prior work for many of our background vdefinitions. Although in [7] it was shown that the attempt-based semantics of this earlier work can be extended to groups of agents, we will use the symbols α, β, γ, δ, and τ to represent single agents in the following definitions.

We use a modal language with the usual connectives of a first order language with equality, as well as operators for propositional attitudes and event sequences. Full details of this modal language and semantics can be found in [2][3]. In summary, BEL has weak S5 semantics and GOAL has system K semantics, with the possible-world model definitions of [2]. KNOW is defined as true knowledge, i.e., (p $\wedge$ (BEL τ p)). (HAPPENS *a*) and (DONE *a*) say that a sequence of actions described by the action expression *a* will happen next or has just happened, respectively. (HAPPENS τ *a*) and (DONE τ *a*) also specify the agent for the action sequence that is going to happen or has just happened. BEFORE and AFTER are defined in terms of HAPPENS. (UNTIL q p) says p will remain true at least until q is true. (PRIOR p q) says that proposition p will become true no later than proposition q. An action expression is built from variables ranging over sequences of events using constructs of dynamic logic: a;b is action composition and *p*? is a test action. We treat BMB between two agents as a semantic primitive in this paper, as in [7]. Mutual belief between two agents α and β is defined in terms of unilateral mutual belief as (BMB α β p) $\wedge$ (BMB β α p) [3]. In our model, BMB can be established by default [1][6].

Definition 1. *PGOAL (Persistent Goal)*

(PGOAL τ p q) $\equiv$
	(BEL τ $\neg p$) $\wedge$
		(GOAL τ $\Diamond p$) $\wedge$

$$\text{(KNOW } \tau \text{ (UNTIL [(BEL } \tau\, p) \vee \text{(BEL } \tau \text{ Y}\neg p) \vee \text{(BEL } \tau\, \neg q)] \\ \text{[GOAL } \tau\, \Diamond p] \text{)) .}$$

Persistent goal formalizes the notion of commitment. An entity τ having a persistent goal *p* is committed to that goal. The entity τ cannot give up the goal that *p* is true in the future, at least until it believes that one of the following is true: *p* is accomplished, or is impossible, or the relativizing condition *q* is untrue. In this paper, we often leave the *q* term unspecified when it has the value of the constant *true*.

Definition 2. *INTEND (Intention)*

$$\text{(INTEND } \tau\, a\, q) \equiv \text{(PGOAL } \tau \text{ [HAPPENS } \tau \text{ (BEL } \tau \text{ (HAPPENS } a))?;a]\ q).$$

Intention to do an action *a* is a commitment to do the action knowingly. The entity τ is committed to being in a mental state in which it has done the action *a* and, just prior to which, it believed that it was about to do the intended action next.

Definition 3. *ATTEMPT*

$$\begin{aligned} &\text{(ATTEMPT } \tau\, e\, \Phi\, \Psi\, t) \equiv \\ &\quad t?;[\text{(BEL } \tau\, \neg\Phi\ ) \wedge \\ &\qquad \text{(GOAL } \tau \text{ (HAPPENS } e;\Diamond\Phi\ ?)) \wedge \\ &\qquad \text{(INTEND } \tau\ \ t?;e;\Psi?\ \text{(GOAL } \tau \text{ (HAPPENS } e;\Diamond\Phi\ ?)))]?;e\ . \end{aligned}$$

An attempt to achieve Φ via Ψ is a complex action expression in which the entity τ is the actor of event *e* at time *t* and, just prior to *e*, the actor chooses that Φ should eventually become true and intends that *e* should produce Ψ relative to that choice. So, Φ represents some ultimate goal that may or may not be achieved by the attempt, while Ψ represents what it takes to make an honest effort.

Definition 4. *PWAG (Persistent Weak Achievement Goal)*

$$\begin{aligned} \text{(PWAG } \tau_1\, \tau_2\, p\, q) \equiv\ & [\neg\text{(BEL } \tau_1\, p) \wedge \text{(PGOAL } \tau_1\, p)] \vee \\ & [\text{(BEL } \tau_1\, p) \wedge \text{(PGOAL } \tau_1 \text{ (MB } \tau_1\, \tau_2\, p))] \vee \\ & [\text{(BEL } \tau_1 \text{ Y}\neg\ p) \wedge \text{(PGOAL } \tau_1 \text{ (MB } \tau_1\, \tau_2 \text{ Y}\neg\ p))] \vee \\ & [\text{(BEL } \tau_1\, \neg q) \wedge \text{(PGOAL } \tau_1 \text{ (MB } \tau_1\, \tau_2\, \neg q))]\ . \end{aligned}$$

This definition, adapted from [11], states that an entity τ_1 has a PWAG with respect to another entity τ_2 when the following holds: (1) if entity τ_1 does not believe that *p* is currently true, it will have a persistent goal to achieve *p*, (2) if it believes *p* to be either true, or to be impossible, or if it believes the relativizing condition *q* to be false, then it will adopt a persistent goal to bring about the corresponding mutual belief with entity τ_2.

Definition 5. *SINCERE*

$$\begin{aligned} \text{(SINCERE } \alpha\, \beta\, p) \equiv \forall e\ \ & \text{(GOAL } \alpha \text{ (HAPPENS } e;\text{(BEL } \beta\, p)?)) \supset \\ & \text{(GOAL } \alpha \text{ (HAPPENS } e;\text{(KNOW } \beta\, p)?))\ . \end{aligned}$$

Entity α is sincere with respect to entity β and proposition *p* if, whenever α wants β to come to believe *p*, it wants β to come to know *p*. Agents may not in fact be sincere but, as with all the mental states discussed here, they are responsible for being sincere.

Definition 6. *TRUST*

$$(\text{TRUSTS } \alpha\ \beta\ p) \equiv (\text{BEL } \alpha\ (\text{BEL } \beta\ p)) \supset (\text{BEL } \alpha\ p)\ .$$

Entity α trusts entity β for proposition *p* if whenever α believes that β believes *p*, α also believes *p*.

Definition 7. *HAPPENING*

$$\begin{aligned}(\text{HAPPENING } a) \equiv\ & (\text{DONE } a) \vee \\ & (\text{HAPPENS } a) \vee \\ & [\exists\, e\ (e \leq a\ ) \wedge (\text{DONE } e) \wedge \neg(\text{DONE } a)]\ .\end{aligned}$$

An action expression *a* is happening if one of the following is true (1) *a* has just been done, or (2) *a* is going to happen next (i.e. *a* is just starting), or (3) there exists some initial subsequence of *a* (represented by *e*) that has just been done but *a* is not yet done.

2.1 REQUEST

We use the single-agent version of the definition of the REQUEST performative that is defined in [7]. Here, α is the entity performing the REQUEST, γ is the intended recipient (the intended actor), *e* is the event of performing the REQUEST, *a* is the action to be done, *q* is a relativizing condition, and *t* is the time point of the utterance.

Definition 8. *REQUEST*

$$\begin{aligned}&(\text{REQUEST } \alpha\, \gamma\, e\ a\ q\ t) \equiv (\text{ATTEMPT } \alpha\ e\ \Phi\ \Psi\ t)\\ &\text{where } \Phi = [(\text{DONE } \gamma\, a] \wedge \\ &\qquad [\text{PWAG } \gamma\, \alpha\ (\text{DONE } \gamma\, a)\ (\text{PWAG } \alpha\, \gamma\ (\text{DONE } \gamma\, a)\ q)]\\ &\text{and } \Psi = [\text{BMB } \gamma\, \alpha\ (\text{BEFORE } e\\ &\qquad\qquad [\text{GOAL } \alpha\\ &\qquad\qquad\qquad (\text{AFTER } e\\ &\qquad\qquad\qquad [\text{PWAG } \alpha\, \gamma\, \Phi\ q]\ )\ ]\ )\ ]\ .\end{aligned}$$

In brief, this definition says that in making a request of addressee γ, the requestor α is trying to get γ to do the action *a*, and to form the commitment to do *a* relative to the requester's commitment that it do it. More formally, by substituting for Φ and Ψ in the definition of ATTEMPT (Definition 3), we obtain the goal and the intention of the REQUEST respectively. The goal of REQUEST is that the intended actor γ eventually do the action *a* and also have a PWAG with respect to the requester α to do *a*. The intended actor's PWAG is with respect to the requester's PWAG (towards γ) that γ does the action *a*. The requester's PWAG is itself relative to some higher-level goal or condition *q*. The intention of REQUEST is that the recipient γ believes there is a mutual belief between the recipient and the requester that before performing the REQUEST the requester α had a goal that, after performing the REQUEST, α will have a PWAG with respect to the intended actor γ about the goal Φ of the request.

2.2 INFORM

The definition of the INFORM speech act that we use within this paper is shown below (derived from [12]). Here, α is the entity performing the INFORM, γ is the intended recipient, *e* is the event of performing the INFORM, *p* is the proposition being informed, and *t* is the time point of the utterance.

Definition 9. *INFORM*

$$
\begin{array}{l}
(\text{INFORM}\ \alpha\ \gamma\ e\ p\ t) \equiv (\text{ATTEMPT}\ \alpha\ e\ \Phi\ \Psi\ t) \\
\quad \text{where } \Phi = [\text{BMB}\ \gamma\ \alpha\ p] \\
\quad \text{and } \Psi = [\text{BMB}\ \gamma\ \alpha \\
\qquad\qquad\qquad (\text{BEFORE}\ e \\
\qquad\qquad\qquad\quad [\text{GOAL}\ \alpha \\
\qquad\qquad\qquad\qquad (\text{AFTER}\ e \\
\qquad\qquad\qquad\qquad\quad [\text{BEL}\ \gamma \\
\qquad\qquad\qquad\qquad\qquad (\text{BEFORE}\ e \\
\qquad\qquad\qquad\qquad\qquad\quad [\text{BEL}\ \alpha\ p]\ )\ ]\ )\ ]\ )\ .
\end{array}
$$

In the definition of INFORM, the sender α has the goal that the intended recipient γ come to believe that there is mutual belief about *p*. The intention of INFORM is that the recipient γ believes there is a mutual belief between the recipient and the informer that before performing the INFORM, the informer α had a goal that after performing the INFORM, the intended recipient γ would believe that, before performing the INFORM, α believed proposition *p*.

3 PROXYING

The notion of proxying involves one entity's asking another entity to do something on its behalf and typically taking responsibility for the action it was asked to do. For this paper, we define two speech acts, PROXY and PROXY-WEAK, that facilitate agents asking other agents to perform speech acts on their behalf. As we will show, the PROXY speech act imposes significant commitments upon the intermediate agent, while the PROXY-WEAK speech act greatly reduces the burden placed upon this proxying agent. Both PROXY and PROXY-WEAK are speech acts based upon REQUEST that take a speech act as an argument – an example of composability of speech acts.

3.1 NOTATION

In the definitions to follow, we need to specify how rewriting occurs for embedded speech acts so we first introduce some notation. In our discussions, we use the following schematic variables: *sact* ranges over speech act types, α is the performer of the speech act, γ is the intended recipient of the speech act, δ is the intended recipient of the speech act performed by the middle agent, *e* is an event type, *a* is an action, *q* is a relativizing condition, and *t* refers to a time point.

We use a parameter substitution function that, when applied to a speech act, replaces all occurrences of the schematic variable representing the specified speech act parameter by the given value. For the speech acts defined within this paper, we use the following abbreviations for speech act parameters: sender (s), intended-recipient (i), distribution (final) recipient (d), event (e), action (a), proposition (p), constraint condition (c), relativizing condition (q), and time (t).

For example, if

$$sact = (\text{INFORM } \alpha\ \gamma\ \text{e } \mathit{on\text{-}vacation}(\alpha)\ \ \text{t})$$

we can specify a new speech act *sact*' using our substitution function

$$sact' = (sact\ \text{s}/\gamma\ \text{i}/\delta\ \text{e}/e'\ \text{t}/t')$$

which represents the original INFORM speech act with all occurrences of the sender parameter replaced by γ, all occurrences of the intended recipient parameter replaced by δ, etc. The parameter substitution function is position independent since the same parameter may occur in different positions in the various speech acts' parameter lists. In such an expression, all unreferenced speech act parameters are left unchanged. In the definitions below in which we use this function, substitution is performed automatically and parameters specified within the embedded speech act do not need to be specified by the uttering agent.

3.2 PROXY

When performing a PROXY speech act, the sender α wants the intended recipient γ to perform the embedded speech act to δ. This descriptive definition is very similar to that for the FIPA PROXY communicative act [5], which states, "The sender wants the receiver to select target agents denoted by a given description and to perform an embedded message to them."[2] The PROXY speech act lends itself to the deployment of middle agents within multiagent domains that can be a fully responsible proxy for other agents. As we will show, we have defined PROXY in such a manner that the

[2] We reproduce FIPA's definition here for convenience but we do not have space in which to explain all of the syntax:

$<i, \text{proxy}(j, \mathit{Ref}\ x\ \delta(x), <j, \mathit{cact}>, \phi)> \equiv$

$<i, \text{inform}(j, \text{I}_i((\exists \text{y})(\text{B}_j(\mathit{Ref}\ x\ \delta(x) = \text{y}) \wedge \text{Done}(<j, \mathit{cact}(\text{y})>, \text{B}_j\phi))))>$

FP : $\text{B}_i\ \alpha \wedge \neg\text{B}_i\ (\text{Bif}_j\ \alpha \vee \text{Uif}_j\ \alpha)$

RE : $\text{B}_j\ \alpha$

where $\alpha = \text{I}_i((\exists \text{y})\ (\text{B}_j(\mathit{Ref}\ x\ \delta(x) = \text{y}) \wedge \text{Done}(<j, \mathit{cact}(\text{y})>, \text{B}_j\phi)))$

individual proxying the embedded communicative act must conform to its logical preconditions.

In the definition of PROXY below, *sact* is an embedded speech act that α wants γ to perform to δ and *c* is a constraint condition for distributing the embedded communicative act (*e.g.* a time deadline).

Definition 10. *PROXY*

(PROXY α γ *e* δ *c* (*sact* s/γ i/δ) *q t*) ≡ (REQUEST α γ *e* (*c*?; (*sact* s/γ i/δ)) *q t*)

PROXY is defined as a request by the sender α for an intermediary entity γ to perform a specified speech act to a final target entity δ if the condition c is met. *sact* is any speech act but the performer of *sact* will be γ and the final recipient will be δ (these substitutions are shown explicitly above but do not need to be performed by the original utterer, α). In the definition above, *e* is the event of the PROXY action at time *t* and *q* is the relativizing condition of the embedded REQUEST.

An example of PROXY in use is shown below (assume a FIRE performative with intuitively defined semantics – by its utterance, the receiver is fired), where Nixon tells Richardson to fire Cox. If Richardson honors Nixon's PROXY, Cox will be fired and Richardson will be the party responsible for Cox's firing.

```
(PROXY
 s/Nixon
 i/Richardson
 e/e
 d/Cox
    (FIRE
     s/Richardson
     i/Cox
     e/e'
     t/t'
    )
 q/true
 t/t
)
```

We can now establish the following results about the mental states of the middle agent.

Theorem 1a: After the middle agent γ honors a PROXY of a REQUEST to do action *a*, it becomes committed to the final recipient doing action *a*. Formally,

|= (DONE (PROXY α γ *e* δ *c* SACT *q t*);*c*?;SACT)
  ∧ (SINCERE γ δ [PWAG γ δ Φ *q*])
  ⊃ (PGOAL γ (DONE δ *a*) *Q*)

where,

SACT = (REQUEST γ δ *e' a q' t'*),
Q is the relativizing condition defined below,

and Φ is the goal of the PROXY (definitions 10, 8).

Proof sketch: The middle agent γ has just honored the proxy. Therefore, from the antecedent, (DONE (REQUEST γ δ *e'* *a* *q'* *t'*)) is true. From the definition of REQUEST as an ATTEMPT (Definition 8), the intention of this REQUEST is Ψ, where:

Ψ = [BMB δ γ
(BEFORE *e'*
[GOAL γ
(AFTER *e'*
[PWAG γ δ Φ' *q*])])]

Φ' = (DONE δ *a*) $\wedge$ *P* ,

and P represents the PWAG conjunct in Φ above of PROXY (Definition 8, 10).

From the definition of ATTEMPT (Definition 3), we see that (INTEND γ t?;*e'*;Ψ? …) must have been true just before γ did the REQUEST action. In other words, γ must have had an intention to bring about a BMB between the recipient and itself that before γ made the request γ had the goal that after the REQUEST is done, it will have a PWAG with the final recipient δ about Φ'. Assuming that agents are sincere in their communication, γ must have the PWAG with final recipient δ about Φ' after it does the REQUEST action because sincere agents cannot intend to bring about a BMB about a proposition they believed to be false. Therefore, (PWAG γ δ Φ' *q*) is true after the REQUEST *e'* is done. Since γ has just done the REQUEST action, it does not yet believe that the final recipient δ has done the action *a*. That is, $\neg$(BEL γ (DONE δ *a*) $\wedge$ *P*) is true. Therefore, from the definition of PWAG (Definition 4), we see that the first disjunct

[$\neg$(BEL γ *p*) $\wedge$ (PGOAL γ *p*)]

is true, where

p = (DONE δ *a*) $\wedge$ *P*

Substituting for *p* in the PGOAL conjunct above, we get

(PGOAL γ (DONE δ *a*) $\wedge$ *P*)

By definition, if an agent is committed to the conjunction *p*1 $\wedge$ *p*2, it must be committed to each of *p*1 and *p*2 relativized to the original commitment. Therefore,

(PGOAL γ (DONE δ *a*) $\wedge$ *P*) $\supset$ (PGOAL γ (DONE δ *a*) *Q*)

where, *Q* = (PGOAL γ (DONE δ *a*) $\wedge$ *P*)

This proves the desired result. Y

The ramification of Theorem 1a is that the middle agent γ of a PROXY personally acquires not only a commitment towards α to perform the embedded REQUEST, but a commitment towards the final agent δ as well. This imposes a significant responsibility upon the middle agent and is not something that all middle agents will wish to accept. Later in this paper we will introduce the PROXY-WEAK speech act that greatly reduces the responsibilities of the middle agent in the case of embedded REQUEST actions.

Theorem 1b: Just before middle agents γ honor a PROXY of an INFORM for some proposition p, they are required to believe p. Formally,

$$\begin{aligned} &\models (\text{DONE } (\text{PROXY } \alpha\, \gamma\, e\, \delta\, c \text{ SACT } q\ t);c?;\text{SACT}) \\ &\qquad \wedge\ (\text{SINCERE } \gamma\, \delta\, p) \\ &\quad \supset\ (\text{BEFORE } e'\ [\text{BEL } \gamma\, p]) \\ &\text{where,} \\ &\quad \text{SACT} = (\text{INFORM } \gamma\, \delta\, e'\ p\ t') \end{aligned}$$

Proof sketch: We use similar arguments as in the proof of Theorem 1a. γ has just honored the PROXY by performing the embedded INFORM. From the definition of INFORM as an ATTEMPT (Definition 9), the intention part of INFORM is

$$\begin{aligned} \Psi = [&\text{BMB } \delta\, \gamma \\ &\quad (\text{BEFORE } e' \\ &\qquad [\text{GOAL } \gamma \\ &\qquad\quad (\text{AFTER } e' \\ &\qquad\qquad [\text{BEL } \delta \\ &\qquad\qquad\quad (\text{BEFORE } e' \\ &\qquad\qquad\qquad [\text{BEL } \gamma\, p]\ )\ ]\ )\ ]\ )\ ] \end{aligned}$$

Since the INFORM has just been done, the middle agent γ must have had the intention to bring about BMB that γ believed p before performing the INFORM. Therefore, by the sincerity assumption, γ must have believed p, i.e., (BEFORE e' [BEL $\gamma\, p$]) is true. This proves the desired result. Y

Honoring a PROXY of an embedded INFORM imposes a significant responsibility upon the middle agent such that if the middle agent cannot verify the proposition's truth value it simply cannot honor the PROXY from α. While there are many situations where the middle agents can satisfy such a strong requirement there are also many situations where the middle agent should not be forced and cannot be expected to believe the embedded proposition. We believe that we can accommodate both situations with our semantics. The PROXY-WEAK speech act that we introduce below provides more options for the middle agent as it removes this strong responsibility to believe the proposition.

3.3 PROXY-WEAK

Next, we define a form of proxy that we call PROXY-WEAK that removes the "strong" requirement of precondition conformance upon the intermediate agent γ and, upon satisfaction and successful performance, provides third-party speech act semantics. Unlike PROXY of a REQUEST to do an action a, the PROXY-WEAK of a REQUEST should not commit the middle agent to the final recipient doing action a. And, unlike PROXY of an INFORM for proposition p, the PROXY-WEAK of an INFORM should not require the middle agent to believe p.

Perhaps most importantly, PROXY-WEAK should support the requirements of a third-party performative [3] – the successful execution of the PROXY-WEAK and subsequent embedded speech act should be equivalent to the sender's performing a speech act directly to the final agent, *even when going through the proxy*. We start with a definition corresponding essentially to FIPA's definition of PROXY-WEAK

[5][3] and, after finding that it has significant limitations, define a version that we believe captures the key missing aspects.

Definition 11a. *PROXY-WEAK* (incorrect)

(PROXY-WEAK $\alpha\,\gamma\,e\,\delta\,c$ sact $q\,t) \equiv$
(REQUEST $\alpha\,\gamma\,e$
$[c?;$(INFORM $\gamma\,\delta\,e'$ (GOAL $\alpha\,\Diamond$(DONE γ sact)) $t')]\ q\ t)$

for sender α, intended (proxying) recipient γ, event e, final target δ, condition c, relativizing condition q, and time t. Furthermore, sact may be any speech act, but the sender will be γ and the final recipient will be δ, i.e. sact = (*sact* s/γ i/δ).

Essentially, this definition of PROXY-WEAK has the originating agent α saying to the intermediate agent γ, "When c is true, perform the INFORM to δ regarding my wanting you to perform the indicated speech act". The middle agent γ then is supposed to say to δ, "α wants me to do *sact* to you."

Since the middle agent γ always perform an INFORM in honoring PROXY-WEAK, from Theorem 1b the following is true:

(BEFORE e'
[BEL γ
(GOAL α
$\Diamond$[DONE γ (*sact* s/γ r/β i/δ)])])

That is, before performing the INFORM to δ, γ must believe that α wanted it to perform *sact*. However, by performing a PROXY-WEAK, the goal of α was *not* that γ does *sact*, but rather that γ perform an INFORM regarding the *sact*. This misrepresents α's goals to δ and is therefore incorrect.

The above definition also does not result in performance of a third-party performative by the proxying agent. To illustrate this point, consider the Nixon example given earlier. Suppose Nixon (α) performs the equivalent of a PROXY-WEAK to Richardson (γ) with *sact* being the performative for 'fire' and Cox being the target agent (δ). According to the above definition of PROXY-WEAK, Richardson can satisfy Nixon's PROXY-WEAK by performing an INFORM to Cox corresponding in natural language to Richardson's saying to Cox, "Nixon wants me to fire you". However, this INFORM does not result in Cox's getting fired by Nixon. The key here is that performatives are accomplished in virtue of their being uttered and here Richardson's utterance does not result in 'fire' being performed by Nixon. The next definition addresses this limitation.

[3] We reproduce FIPA's definition here for convenience but we do not have space in which to explain all of the syntax:
$<i$, proxy(j, *Ref* x $\delta(x)$, <j inform<y, I_i Done($<i$, cact(y)>))>, ϕ)> $\equiv$
$<i$, inform(j, $I_i((\exists y)(B_j($*Ref* x $\delta(x) = y) \wedge$ Done(<j inform<y, I_i Done($<i$, cact(y)>))>, $B_j\,\phi$))))>
FP : $B_i\,\alpha \wedge \neg B_i\,(\mathrm{Bif}_j\,\alpha \vee \mathrm{Uif}_j\,\alpha)$
RE : $B_j\,\alpha$
where $\alpha = I_i((\exists y)\,(B_j($*Ref* x $\delta(x) = y) \wedge$ Done(<j inform<y, I_i Done($<i$, cact(y)>))>, $B_j\,\phi$)))

Definition 11b. *PROXY-WEAK*

(PROXY-WEAK $\alpha\,\gamma\,e\,\delta\,c\,sact\,q\,t) \equiv$

(REQUEST $\alpha\,\gamma\,e\,[c?;(\text{INFORM}\,\gamma\,\delta\,e'\,\theta\,t')]\,q\,t)$

where,

θ = (HAPPENING sact) and

sact = (*sact* s/α i/δ e/*e;e*' t/*t*')

In other words, PROXY-WEAK of a speech act *sact* is a REQUEST to INFORM that the speech act *sact* is happening using the two acts – the sender's and the intermediary's. The two actions are *e;e*', where *e*' is the very act of informing this fact – hence γ's act of performing the INFORM also completes α's speech act to δ.

Using this definition of PROXY-WEAK, Richardson *will* satisfy Nixon's PROXY-WEAK by saying, in natural language, "Nixon hereby fires you". Here 'fires' is used as a third party performative – it is a performative because saying so in the right situation makes it so. We note that by the definition of PROXY-WEAK as a REQUEST, when the middle agent γ accepts the REQUEST, it has a PWAG with sender α about performing the INFORM act with respect to the sender's PWAG that γ does the INFORM. From the definition of PWAG (Definition 4), γ will establish a mutual belief to that effect after performing the requested INFORM. The PROXY-WEAK is discharged successfully when this mutual belief is established. This is evident from the next two theorems.

Theorem 2a: When the middle agent successfully discharges a PROXY-WEAK performed to it, the original sender believes that it has performed the embedded speech act to the target even though it may not have observed the middle agent's act directly and only knows that it was done. Formally,

$$\begin{aligned}&\models [\text{DONE}\ (\ [\text{PROXY-WEAK}\ \alpha\,\gamma\,e\,\delta\,c\,SACT\,q\,t];\\ &\qquad\qquad [\text{MB}\ \alpha\,\gamma\,(\text{DONE}\ [\text{INFORM}\ \gamma\,\delta\,e'\ (\text{HAPPENING}\ SACT)\ t']\ )\ ]?\\ &\qquad\qquad)\ ]\\ &\quad \wedge (\text{SINCERE}\ \alpha\,\gamma\,[\text{PWAG}\ \alpha\,\gamma\,\Phi\,q])\\ &\quad \supset (\text{BEL}\ \alpha\ (\text{DONE}\ \alpha\ SACT))\end{aligned}$$

where,

SACT = (*sact* s/α i/δ e/*e;e*' t/*t*') and

Φ is the goal of the PROXY-WEAK (Definitions 11b, 8).

Proof sketch: By performing a PROXY-WEAK, the sender α requested the middle agent γ to inform the final recipient δ that (HAPPENING *SACT*). (1) From the usual assumption of sincerity, α cannot make that REQUEST unless α believes that *p*. (2) When γ establishes the mutual belief that the INFORM has been done, α believes that the event *e*' (i.e. the INFORM event) has been done. From (1) and (2), α believes that the event *sequence e;e*' is happening and also believes that the event *e*' has just been done. So α believes that the event sequence *e;e*' has just been done and hence believes that the action *SACT* represented by the event sequence *e;e*' has just been done. This establishes the desired result. Y

Theorem 2b: When the middle agent satisfies a PROXY-WEAK performed to it, the final recipient will come to believe that the original sender has performed the embedded speech act to it provided that it trusts the middle agent. Formally,

$$\begin{aligned}&\models [\text{DONE (PROXY-WEAK } \alpha\, \gamma\, e\, \delta\, c\, \mathit{SACT}\, q\, t);c?;(\text{INFORM } \gamma\, \delta\, e'\, p\, t')]\\ &\quad\wedge (\text{TRUSTS } \delta\, \gamma\, p)\\ &\quad\wedge (\text{SINCERE } \gamma\, \delta\, p)\\ &\quad\supset (\text{BEL } \delta\, (\text{DONE } \mathit{SACT}))\end{aligned}$$

where

$\mathit{SACT} = (\mathit{sact}\ \text{s}/\alpha\ \text{i}/\delta\ \text{e}/e;e'\ \text{t}/t')$ and
$p = (\text{HAPPENING } \mathit{SACT})$

Proof sketch: By assumption of sincerity, the middle agent γ believes the proposition being informed. The final recipient δ trusts the middle agent γ. (1) Therefore, δ also believes the proposition being informed i.e. δ believes that *p*. (2) The final recipient δ has just received the INFORM from the middle agent. Therefore, δ believes that the event *e*' (i.e. the INFORM event) has been done. From (1) and (2), δ believes that the event sequence *e;e*' is happening and also believes that the event *e*' has just been done. So it believes that the event sequence *e;e*' has just been done and hence believes the action *SACT* represented by the event sequence *e;e*' has just been done. This establishes the desired result. Y

Theorem 3a: After a middle agent honors a PROXY-WEAK of a REQUEST to do action *a*, it does *not* become committed to the final recipient doing action *a*. Formally,

$$\begin{aligned}&\not\models [\text{DONE (PROXY-WEAK } \alpha\, \gamma\, e\, \delta\, c\, \text{SACT}\, q\, t);c?;(\text{INFORM } \gamma\, \delta\, e'\, \theta\, t'))\\ &\quad\supset (\text{PGOAL } \gamma\, (\text{DONE } \delta\, a)\, Q)\end{aligned}$$

where,

$\text{SACT} = (\text{REQUEST } \alpha\, \delta\, e'\, a\, q'\, t')$,
$\theta = (\text{HAPPENING SACT})$, and
Q is a relativizing condition.

Proof sketch: From definition 11b, note that the middle agent γ performs an INFORM • γ never performs the embedded speech act no matter what it is. Therefore, when *sact* is a REQUEST, γ does not have the goal and intentions of (REQUEST α δ *e*' *a q*' *t*'). In particular, from the definition of INFORM (Definition 9), γ does not have a PWAG with δ for doing *a* and hence is not committed to δ doing *a*. The relativizing condition *Q* does not come into play and, without loss of generality, may have any value other than false. Y

Theorem 3b: After the middle agent honors a PROXY-WEAK of an INFORM for some proposition *p*, it is *not* required to have believed *p*. Formally,

$$\begin{aligned}&\not\models [\text{DONE (PROXY-WEAK } \alpha\, \gamma\, e\, \delta\, c\, \text{SACT}\, q\, t);c?;(\text{INFORM } \gamma\, \delta\, e'\, \theta\, t')\ ]\\ &\quad\supset (\text{BEFORE } e'\ [\text{BEL } \gamma\, p]\ )\end{aligned}$$

where,

$\text{SACT} = (\text{INFORM } \alpha\, \delta\, e'\, p\, t')$ and
$\theta = (\text{HAPPENING SACT})$

Proof sketch: This is similar to the proof of Theorem 3a, where *sact* is an INFORM. In this case, the middle agent γ performs an INFORM with the propositional content of (HAPPENING *sact*) rather than an INFORM with the propositional content *p* directly. The middle agent γ therefore must believe (HAPPENING *sact*) just prior to honoring the PROXY-WEAK, but not necessarily *p*. Y

4 Discussion

Our analysis of the semantics of PROXY and PROXY-WEAK above shows that middle agents can be deployed in multiagent systems and flexibly accommodate a wide range of domains and scenarios, some in which the middle agents must take full responsibility for their actions and some in which the middle agents act more as if they were simple couriers. Because of the strong semantic definitions involved, an agent faced with a decision to perform a PROXY or a PROXY-WEAK speech act can reason about the burden placed upon the middle agent and choose between them knowingly. Similarly, a middle agent receiving one of these speech acts can reason about the level of responsibility expected of it and make a knowledgeable decision about whether to honor the PROXY or PROXY-WEAK. We can demonstrate through a set of proofs similar to that used in [7] that these semantics will also hold for groups of agents as targets of PROXY, PROXY-WEAK, and the embedded speech acts.

Prior work on agent communication languages (e.g., [4][5][9][10]) either lack support for middle agent speech acts or lack the strength and depth of semantics as we have introduced above. Furthermore, there has been no work on agent communication languages that has successfully defined a speech act that supports third-party semantics until now. The prior work most similar to that presented within this paper has been performed by FIPA [5]. The FIPA standards body has defined proxy communicative acts with an intent similar to ours [5]. There are several significant differences between our approach and that of FIPA, however. First, FIPA's PROXY is defined as an INFORM between the originating agent and the middle agent, while ours is defined using a REQUEST to the middle agents. This is significant in that the middle agent within FIPA need not be expected to do anything, while in our definition the middle agent is expected to perform a subsequent speech act if it agrees to honor the REQUEST. Second, FIPA defines their equivalent of PROXY-WEAK in terms of an INFORM of an intent by the original sender to have the middle agent do the embedded communicative act. Because the middle agent will only ever perform an INFORM and never the embedded speech act directly, the FIPA definition therefore misrepresents the sending agent's intentions to the target agent (see the discussion of Definition 11a).

In summary, the PROXY and PROXY-WEAK communicative acts defined in this paper provide speech acts that support agents interacting with middle agents that can act on their behalf. Our analysis has shown that the two acts result in the middle agents having significantly different levels of commitments relative to the final group, where PROXY imposes significant and PROXY-WEAK imposes very little responsibility upon the middle agents. We also have shown that the PROXY-WEAK speech act results in the correct embodiment of third party performative semantics, where we obtain the equivalence of the sending agents performing a speech act directly on the final target agents even while going through proxies.

Acknowledgements

We gratefully acknowledge the DARPA CoABS Program (contract F30602-98-2-0098, A0 G352) for supporting the research presented in this paper. Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC06-76RLO 1830. We would also like to thank the anonymous reviewers for their excellent comments.

References

[1] Cohen, P.R. On Knowing What to Say: Planning Speech Acts. *Ph.D. Thesis*, Department of Computer Science, University of Toronto, 1978.
[2] Cohen, P.R., and Levesque, H.J. Intention is Choice with Commitment, *Artificial Intelligence*, 42(3), 1990.
[3] Cohen, P.R., and Levesque, H.J. Performatives in a Rationally Based Speech Act Theory. *Proc. of the 28th Annual Meeting of the Association for Computation Linguistics*, 1990.
[4] Finin, T., and Labrou, Y. A proposal for a new KQML Specification, TR CS-97-03, CSEE Dept., UMBC, Baltimore, MD, 1997.
[5] FIPA ACL Specification. http://www.fipa.org/, 2000.
[6] Katagiri Y. Belief Coordination by Default. *Proceedings of the 2nd International Conference on Multi-Agent Systems*, pages 142-149, 1996.
[7] Kumar, S., Huber, M.J., McGee D.R., Cohen, P.R., and Levesque, H.J. Semantics of Agent Communication Languages for Group Interaction. *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI)*, pages 42-47, 2000.
[8] Martin, D.L., Cheyer, A.J., and Moran, D.B., The Open Agent Architecture: A Framework for Building Distributed Software Systems, *Applied Artificial Intelligence*, vol. 13, pages 91-128, 1999.
[9] Pitt J. and Mamdani. A.. Designing Agent Communication Languages for Multi-Agent Systems. In F. Garijo and M. Boman (eds.): *Multi-Agent System Engineering: Proceedings MAAMAW'99*, LNAI1647, Springer-Verlag, pp102–114, 1999.
[10] Singh, M.P. A Social Semantics for Agent Communication Languages. *Proceedings of the IJCAI 2000 Workshop on Agent Communication Languages*, 2000.
[11] Smith, I.A., and Cohen, P.R. Toward a Semantics for an Agent Communications Language Based on Speech-Acts. *Proceedings of the 13th National Conference on Artificial Intelligence*, 1996.
[12] Smith, I.A., Cohen, P.R., Bradshaw, J. M., Greaves, M., and Holmback, H. Designing Conversation Policies Using Joint Intention Theory. *Proceedings of the 3rd International Conference on Multi-Agent Systems,* 1998.

Commitment Machines[⋆]

Pınar Yolum and Munindar P. Singh

Department of Computer Science,
North Carolina State University
Raleigh, NC 27695-7535, USA
{pyolum, mpsingh}@eos.ncsu.edu

Abstract. We develop an approach in which we model communication protocols via *commitment machines*. Commitment machines supply a content to protocol states and actions in terms of the social commitments of the participants. The content can be reasoned about by the agents thereby enabling flexible execution of the given protocol. We provide reasoning rules to capture the evolution of commitments through the agents' actions. Because of its representation of content and its operational rules, a commitment machine effectively encodes a systematically enhanced version of the original protocol, which allows the original sequences of actions as well as other legal moves to accommodate exceptions and opportunities. We show how a commitment machine can be compiled into a finite state machine for efficient execution, and prove soundness and completeness of our compilation procedure.

1 Introduction

Protocols are structured interactions among communicating agents. Protocols are essential in applications such as electronic commerce where it is necessary to constrain the behaviors of autonomous agents. Multiagent protocols have traditionally been modeled by formalisms similar to those used to specify protocols in distributed computing and computer networks (e.g., finite state machines [2] or Petri Nets [4]). These formalisms specify protocols merely in terms of legal sequences of actions without regard to the meanings of those actions. Consequently, protocols tend to suffer from unnecessary rigidity in execution [7], resulting in redundant interactions and avoidable failures.

Let us consider some desirable properties of a protocol representation.

– *Autonomy:* Promoting the participants' autonomy is crucial for creating effective systems in open environments. Participants must be constrained in their interactions only to the extent necessary to carry out the given protocol, and no more.
– *Opportunities:* Participants should be able to take advantage of opportunities to improve their choices or to simplify their interactions. Depending on the situation, a participant may take advantage of domain knowledge, and jump to a state in a protocol without explicitly visiting one or more intervening states, since visiting each state may require additional messages and cause delays.

[⋆] We would like to thank James Lester and Peter Wurman for helpful comments. This research was supported by the National Science Foundation under grant IIS-9624425 (Career Award). A preliminary version of this paper appears in the WET ICE 2000 Proceedings.

– *Exceptions:* Participants must be able to modify their interactions to handle exceptions that result from the unexpected behavior of the participants. For example, a deadline may be renegotiated at a discount. This would obviously involve domain knowledge, but the protocol representation should allow it.

We propose *commitment machines (CMs)* as a formalism to formally specify and execute protocols. A commitment machine attaches specific *declarative* meanings to states and actions within a protocol. These meanings are based on commitments of the participating agents. When the meanings of states and actions are formally defined, the legal computations can be logically inferred. This enables the protocols to be systematically enhanced with additional transitions, allowing a broader range of interactions. The enhancement enables agents to exploit opportunities and handle exceptions.

We show how a commitment machine may be automatically compiled into a finite state machine (FSM) in which no commitments or other declarative meanings are explicitly mentioned, but which can be efficiently executed. We give technical results proving that the compilation procedure, sometimes with additional restrictions, produces an FSM that is *deterministic* (easy to execute), *sound* (never produces a computation not allowed by the commitment machine), and *complete* (can produce the effect of any computation allowed by the commitment machine).

The rest of this paper is organized as follows. Section 2 describes our example and necessary background information. Section 3 introduces commitment machines and show how they may be applied. Section 4 shows how commitment machines can be compiled into finite state machines and establishes important results regarding the compilation procedure. Section 5 describes our contributions with respect to the most relevant literature.

2 Technical Framework

Following speech act theory, we view communication as a form of action [1]. The actions here reflect progress in the given protocol and may be captured in terms of modifications to the participants' commitments.

Social commitments are commitments made from one agent to another agent to carry out a certain course of action [3,11]. A social commitment C*(x, y, p)* relates a debtor *x*, a creditor *y*, and a condition *p*. When a social commitment of this form is created, *x* becomes responsible to *y* for satisfying *p*. The condition *p* may involve relevant predicates and commitments, allowing the commitments to be nested or conditional. The commitments are flexible and can be revoked or modified. Almost always, the revocation or modification is constrained through other commitments.

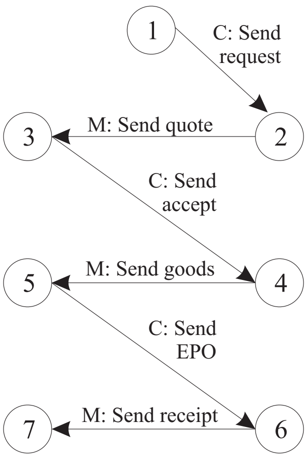
Viewing a commitment as an abstract data type, the creation and the manipulation of the commitments can be described using the following operations [11,14]. Here, x, y, z denote agents, and c and c' denote commitments of the form C*(x, y, p)*.

1. *Create(x, c)* establishes the commitment *c*.
2. *Discharge(x, c)* resolves the commitment *c*, i.e., the condition p starts to hold.
3. *Cancel(x, c)* cancels the commitment *c*. Usually, the cancellation of a commitment is accompanied by the creation of another compensating commitment.

4. *Release(y, c)* releases the debtor from the commitment c. It can be performed by the creditor, to mean that the debtor is no longer obliged to carry out his commitment.
5. *Assign(y, z, c)* eliminates the commitment c, and creates a new commitment c' for which z is appointed as the new creditor.
6. *Delegate(x, z, c)* eliminates the commitment c, and creates a new commitment c' in which the role of the debtor is transferred to z.

As a running example, we consider a simplified version of the NetBill protocol which was developed to handle the buying and selling over the Internet of electronic goods, such as software and electronic documents [12].

Example 1. As shown in the figure below, the protocol begins with a customer requesting a quote for some desired goods, followed by the merchant sending the quote. If the customer accepts the quote, then the merchant delivers the goods and waits for an electronic payment order (EPO). The goods delivered at this point are encrypted, that is, not usable. After receiving the EPO, the merchant sends the receipt to the customer, who can then successfully decrypt and use the goods. Some scenarios that will not be handled by this protocol specification are shown on the right side of the figure.



(i)Instead of waiting for a customer to request a quote, a merchant may proactively send a quote, mimicking the idea of advertising.
(ii) The customer may send an "accept" message without first exchanging explicit messages about a price. This situation would reflect the level of trust the customer places in the merchant.
(iii) A merchant may send the goods without an explicit price quote. Sending unsolicited goods would correspond to "try before you buy" deals, common in the software industry.

Example 2. We define the semantic content of each state in the figure above based on the participants' commitments:

– In state 3, having sent a quote to a customer, the merchant commits to delivering goods and sending a receipt afterwards, if the customer promises to pay.
– In state 4, having sent an accept to a merchant, the customer agrees to pay, but only if the merchant promises to send a receipt afterwards.
– In state 5, the merchant has fulfilled part of his promise by sending the goods.
– In state 6, the customer has discharged his commitment of sending the EPO.
– In state 7, the merchant has discharged his commitment of sending the receipt.

3 Commitment Machines

We define a commitment machine (CM) in terms of sets of states and actions that are given a declarative semantic content in terms of commitments. A CM specifies

- the possible states an executing protocol can be in.
- the actions that are used for a transition from one state to another.
- the possible final states of the protocol.

The meaning associated with each state specifies which commitments are in force in that particular state, and the meaning associated with each action defines how the commitments are affected by that action (thereby leading to a state change).

Like an FSM, a CM has a current state; zero or more actions are allowed in each state; every allowed action causes the CM to transition to a new state. Unlike an FSM, the representation of a CM does not specify a starting state. The participants may start the protocol from a state by accepting the commitments that are in force in that state. Usually, the protocol will have some states where no commitments are in force. A CM also has final states, which reflect the acceptable or desirable termination states of the protocol.

Importantly, unlike in an FSM, the transitions between the states are not explicitly specified. Based on the intrinsic meaning of the actions, the new state that is reached by performing an action at a particular state can be logically inferred. Thus, instead of specifying the sequences of actions that can be performed, a CM simply specifies the meanings that are legal in the protocol and, of these, the meanings that are final.

A CM specification of a protocol emphasizes that the aim of executing the protocol is not merely to perform certain sequences of actions, but to reach a desirable state. With this in mind, we can come up with different action sequences or paths that accomplish the same goal as the original path.

Our formalization is based on a language used to represent the legal meanings in a CM. Our formal language, $\mathcal{P}$, is based on the language of propositional logic with the addition of a commitment operator to represent commitments, and a *leads to* operator to capture strict implication.

The following Backus-Naur Form (BNF) grammar with a start symbol Protocol gives the syntax of $\mathcal{P}$. In this grammar, slant typeface indicates nonterminals; $\longrightarrow$ is a metasymbol of BNF; $\ll$ and $\gg$ delimit comments; $\{$ and $\}$ indicate that the enclosed item is repeated 0 or more times; the remaining symbols are terminals. For expository ease, we use a simplified notation for commitments. Here $\mathsf{C}_x p$ means that x (which can be the customer or the merchant in NetBill) is committed to the other party to carry out p. Further, we restrict the nesting of commitments to one level.

- Protocol $\longrightarrow$ $\{$Action$\}$ $\ll$set of actions$\gg$
- Action $\longrightarrow$ Token: L $\ll$token is a label; L is the associated meaning$\gg$
- Commitment $\longrightarrow$ C_x (L) $\mid$ C_x (M) $\ll$simplified as explained below$\gg$
- L $\longrightarrow$ Commitment
- M $\longrightarrow$ L $\leadsto$ L $\ll$leads to, indicating a strict implication$\gg$
- L $\longrightarrow$ L $\wedge$ L $\ll$conjunction$\gg$
- L $\longrightarrow$ $\neg$ L $\ll$negation$\gg$
- L $\longrightarrow$ Prop $\ll$atomic propositions$\gg$

The boolean operators are given the usual semantics. The strict implication, $p \leadsto q$, requires q to hold when p holds. Contrary to the material implication ($p \rightarrow q$), which is true when p is false, the strict implication is false if p is false. The strict implication is used only in commitments. Our formal semantics is given in Appendix A. For commitments where x and y are the same $(\mathsf{C}_x(p \leadsto \mathsf{C}_x r))$, the simpler form $\mathsf{C}_x(p \leadsto r)$ suffices.

Example 3. Following Figure 1, we define the following atomic propositions and commitments that will be used to specify the meanings.

- **Atomic propositions**
  - request ≪the customer has requested a quote.≫
  - goods ≪the merchant has delivered the goods.≫
  - pay ≪the customer has paid the agreed amount.≫
  - receipt ≪the merchant has delivered the receipt.≫
- **Abbreviations for the commitments**
  - accept ≪an abbreviation for C_c(goods $\leadsto$ pay) meaning that the customer is willing to pay if he receives the goods.≫
  - promiseGoods ≪an abbreviation for C_m(accept $\leadsto$ goods) meaning that the merchant is willing to send the goods if the customer promises to pay.≫
  - promiseReceipt ≪an abbreviation for C_m(pay $\leadsto$ receipt) meaning that the merchant is willing to send the receipt if the customer pays.≫
  - offer ≪an abbreviation for (promiseGoods $\wedge$ promiseReceipt)≫

The meaning of a state is given by any formula derivable from the nonterminal L. We define two logical relations among meanings: logical derivation and equivalence. $p \vdash q$ means that q can be logically derived from p. $p \equiv q$ means that p and q are logically equivalent, that is, $p \vdash q$ and $q \vdash p$. These two relations are used to compare the execution states of a protocol semantically.

A minimal set of meanings is one in which all meanings are logically distinct. A set of final meanings is consistent if it is well-behaved with respect to logical consequence.

Definition 1. A set $\mathbf{M}$ of meanings is *minimal* if and only if the following hold:

- $(\forall m_i, m_j \in \mathbf{M}$: $(m_i \equiv m_j) \Rightarrow (m_i = m_j))$
- $\mathsf{true} \in \mathbf{M}$.
- $\mathsf{false} \notin \mathbf{M}$. ▌

Definition 2. A set of final meanings $\mathbf{F}$ is *consistent* with respect to a set of meanings $\mathbf{M}$ if and only if any meaning that is stronger than a final meaning is also final. That is, $(\forall m_i \in \mathbf{F}, m_j \in \mathbf{M}$: $(m_j \vdash m_i) \Rightarrow (m_j \in \mathbf{F}))$. ▌

Actions are represented as a pair whose first element is the token (name) of the action, and whose second element is the effect of the action. That is, $\langle a : e \rangle$ is an action, if a is the token and e is the meaning (effect) of the action.

Definition 3. A CM is a triple $\langle \mathbf{M}, \Delta, \mathbf{F} \rangle$, where $\mathbf{M}$ is a finite minimal set of meanings, Δ is a finite set of actions defined in terms of commitments, and $\mathbf{F} \subseteq \mathbf{M}$ is a consistent set of final meanings. ▌

Table 1. The CM representation of the NetBill protocol

	Meanings (M)	Actions (Δ)	Final Meanings (F)
1	true	$\langle$sendRequest: request$\rangle$	request
2	request	$\langle$sendQuote: offer$\rangle$	offer
3	offer	$\langle$sendAccept: accept$\rangle$	goods $\wedge$ pay $\wedge$ receipt
4	$\mathsf{C}_m$goods $\wedge$ accept $\wedge$ promiseReceipt	$\langle$sendGoods: goods $\wedge$ promiseReceipt$\rangle$	
5	goods $\wedge$ $\mathsf{C}_c$pay $\wedge$ promiseReceipt	$\langle$sendEpo: pay$\rangle$	
6	goods $\wedge$ pay $\wedge$ $\mathsf{C}_m$receipt	$\langle$sendReceipt: receipt$\rangle$	
7	goods $\wedge$ $\mathsf{C}_c$pay $\wedge$ receipt		
8	goods $\wedge$ pay $\wedge$ receipt		

Next we specify the legal meanings the NetBill protocol execution can be in, the actions the agents can perform and the final meanings that the protocol can end. Since each action can be performed by only one party, we do not specify the performers explicitly. Table 1 gives the CM specification.

A CM transitions from meaning q to meaning r under action $\langle a : e\rangle$ if and only if after applying the effect e on q, r can be logically derived. We formalize the CM transitions as follows: $q \models_{\langle a:e\rangle} r \triangleq (q \wedge e) \vdash r$. Thus, deriving the resulting meaning from a given meaning and action involves computing the logical consequence (that is, the $\vdash$ relation). For the propositional part of the language this is as usual. For commitments, we now present some important rules for reasoning about their consequences. These reasoning rules capture the operational semantics of our approach.

1. A commitment $\mathsf{C}_x p$ ceases to exist when the proposition p becomes true.
2. A commitment $\mathsf{C}_x(p \leadsto r)$ ceases to exist when the proposition p becomes true, but a new base-level commitment $\mathsf{C}_x r$ is created to capture that x has to satisfy the original commitment by bringing about the proposition r.
3. A commitment $\mathsf{C}_x(p \leadsto r)$ ceases to exist when the proposition r holds (before p is initiated), and no additional commitments are created.

Example 4. Consider the metacommitment $\mathsf{C}_m(\text{pay} \leadsto \text{receipt})$, which denotes the commitment that the merchant is willing to send a receipt if the customer pays. After the creation of this metacommitment, the following scenarios may take place:

– The customer pays, making the proposition pay true. In this case, the metacommitment is terminated and a new commitment, $\mathsf{C}_m$receipt, is created in its stead (Reasoning Rule 2). When the merchant actually sends the receipt, i.e., when the proposition receipt becomes true, then the commitment $\mathsf{C}_m$receipt is discharged (Reasoning Rule 1).
– Before the customer pays, the merchant sends the receipt, making the proposition receipt true. In this case, the metacommitment is terminated, but no other commitment is created since the customer did not commit to paying in the first place (Reasoning Rule 3).

The CM specification of a protocol can be applied both at run time and compile time. A CM specification of a protocol gives the states and the effects of performing the various actions. Given a CM, an agent that can process logical formulas can directly execute the CM. In this respect, the choice of actions is a planning problem for each agent. That is, from the possible final states, the agent first decides on the desired final state, and then logically infers a path that will take it from the current state to the desired final state. Effectively, the agent interprets the CM directly at run time. Alternatively, a CM can be compiled into an FSM that abstracts out the meanings of the states and actions. In the next section, we describe this compilation process in detail.

4 Compiling Commitment Machines

A protocol specification that allows the above characteristics can drastically improve the flexibility and thus the quality of the solution provided by agent-based applications. However, the flexibility comes at the price of reasoning with declarative representations at run-time, which can be expensive and may increase the code footprint of the agents who implement such reasoning. Fortunately, it is possible to compile a CM into an FSM so that the desired affect can be obtained without representing and reasoning about declarative meanings at run time.

Before considering the requirements of compiling a CM into an FSM, let us give a formal definition of an FSM and a DFSM.

Definition 4. A finite state machine is a five-tuple, $M = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$, where $\mathbf{S}$ is the set of states, Σ is the input alphabet, $s_0 \in \mathbf{S}$ is the start state, $\mathbf{Q} \subseteq \mathbf{S}$ is the set of final states, and $\delta \subseteq \mathbf{S} \times \Sigma \times \mathbf{S}$ is the transition relation. A finite state machine is deterministic if any action has at most one transition from a state, i.e., $(\forall s, s', s'' \in \mathbf{S}, a \in \Sigma : (s, a, s'), (s, a, s'') \in \delta \Rightarrow s' = s'')$. ∎

For an agent to be able to directly execute the FSM that results from compiling a CM, we seek an FSM that is deterministic and with no irrelevant transitions. Given a CM, the states of the FSM can be generated from the meanings of the CM. However, the execution of the FSM follows the usual regime of state-transition-state without regard to the formulas that exist in CM meanings on which the FSM states are based.

4.1 Compilation Formalized

We now show how a CM can be formally compiled into an FSM. We establish soundness and completeness results regarding our compilation procedure. The compilation procedure can be realized in an automatic tool. Recall that a CM allows multiple starting states, whereas an FSM allows only one. In the compilation below, we show how an FSM can be constructed after choosing a start state for the CM, namely, true.

Procedure 1. Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a CM. Construct an FSM $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ as follows.

- $\mathbf{S} = \mathbf{M}$
- $\Sigma = \{a : \langle a : e \rangle \in \Delta\}$
- $s_0 = \mathsf{true}$
- $\mathbf{Q} = \mathbf{F}$

– δ = $\{\langle m_i, a, m_j\rangle : m_i, m_j \in$ **M**, $\langle a : e\rangle \in \Delta$ and ($m_i \models_{\langle a:e\rangle} m_j, m_i \not\vdash m_j$ and ($\forall$ $m_k \in$ **M**: $m_i \models_{\langle a:e\rangle} m_k \Rightarrow m_j \vdash m_k$))$\}$ ▮

To infer the valid transitions in the FSM, we consider the possible entailments between the meanings. To ensure determinism and efficiency, we constrain the allowed transitions with two major restrictions.

Restriction 1. $\langle m_i, a, m_j\rangle \in \delta$ entails that $m_i \not\vdash m_j$.

If the source meaning already entails the content captured in the target meaning, no transition from the source to the target is necessary. This restriction ensures that the meaning that will be reached is not already captured at the current state, and that the FSM has no transitions that do not add to the content. ▮

Restriction 2. $\langle m_i, a, m_j\rangle \in \delta$ entails that ($\forall$ $m_k \in$ **M**: $m_i \models_{\langle a:e\rangle} m_k \Rightarrow m_j \vdash m_k$).

This is to ensure that if applying an action at a particular meaning entails several possible meanings, then the transition will end in a state that contains the maximal information. Later we will restrict our CMs so that a maximal meaning always exists. ▮

Notice that δ is defined so as to satisfy Restrictions 1 and 2. When an FSM Y is produced from a CM X by Procedure 1, we say that X is *compiled* into Y. A compiled FSM can be directly executed by an agent with a single thread, using only constant space. Theorem 1 establishes this result.

Theorem 1. An FSM produced by compiling a CM according to Procedure 1 is deterministic.

Proof. Let $X = \langle$**M**, Δ, **F**$\rangle$ be a CM compiled into $Y = \langle$**S**, Σ, s_0, **Q**, $\delta\rangle$. Let $\langle m_i, a, m_j\rangle \in \delta$ and $\langle m_i, a, m_k\rangle \in \delta$ be two transitions of Y. From Restriction 2, we know $m_j \vdash m_k$ and $m_k \vdash m_j$, that is, $m_j \equiv m_k$. Then, by Definition 1, $m_j = m_k$. Thus, by Definition 4, Y is deterministic. ▮

The correctness of a compilation procedure must be based on the computations that can result from it. Therefore, we formalize the notion of a computation, how a computation may be generated by a CM, and how a computation may be realized by an FSM. Let $f \in \mathcal{N}$ be a finite ordinal. Let **I** be the set of indices $\{0, 1, \ldots, (f-1)\}$ (**I** is empty when $f = 0$). Let **J** be the set of indices $\{0, 1, \ldots, f\}$.

Definition 5. τ = $\langle m_0, a_0, m_1, a_1, \ldots, a_{f-1}, m_f\rangle$ is a computation if ($\forall i \in$ **I**$, j \in$ **J** : $a_i \in \Sigma$ and $m_j \in$ **M**). Intuitively, the action labels that occur in τ, $\langle a_0, a_1, \ldots, a_{f-1}\rangle$, describe the externally visible part of the computation because these are the actions seen by other agents. ▮

Definition 6. τ = $\langle m_0, a_0, m_1, \ldots, m_f\rangle$ is *generated* by a CM $X = \langle$**M**, Δ, **F**$\rangle$ if and only if $m_f \in$ **F**, and ($\forall i \in$ **I**: $m_i \in$ **M** and ($\exists e_i, \langle a_i : e_i\rangle \in \Delta$: $m_i \models_{\langle a_i:e_i\rangle} m_{i+1}$)). ▮

Definition 7. τ = $\langle m_0, a_0, m_1, \ldots, m_f\rangle$ is *realized* by an FSM $Y = \langle$**S**, Σ, s_0, **Q**, $\delta\rangle$ if and only if $m_0 = s_0$, $m_f \in$ **Q**, and ($\forall i \in$ **I**, $\langle m_i, a_i, m_{i+1}\rangle \in \delta$). ▮

Notice that a computation can only be generated by a machine that can manipulate these meanings, that is, a CM. By contrast, an FSM can realize this computation by following the action sequence. With this distinction in mind, we define two main aspects of correctness. *Soundness* means that only allowed computations are realized. *Completeness* means that all allowed computations can be realized. A compilation procedure that produces an FSM $\langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ with $\mathbf{S}=\{\}$ would be sound, whereas one that produces an FSM with $\Sigma=\mathbf{M}\times\Delta\times\mathbf{M}$ would be complete. That is, it is trivial to ensure either soundness or completeness. However, it is crucial to ensure both properties. Theorem 2 establishes the soundness of our compilation method: it states that the compiled FSM won't produce a computation that was not allowed by the original CM.

Theorem 2. Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be compiled into $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$. Then any computation realized by Y is generated by X.
Proof. Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation realized by Y. By Procedure 1 $\mathbf{S} = \mathbf{M}$, $(\forall i \in \mathbf{I}, m_i \in \mathbf{M})$, and $m_f \in \mathbf{F}$. Consider the ith transition in τ, $\langle m_i, a_i, m_{i+1} \rangle \in \delta$. This implies, $(\exists e_i : \langle a_i : e_i \rangle \in \Delta$ and $m_i \models_{\langle a_i : e_i \rangle} m_{i+1})$. By Definition 6, X generates τ. ■

4.2 Completeness

Interestingly, we must refine the definition of a CM to ensure completeness. First, a computation that is generated by a CM may begin from any arbitrary meaning. Second, there may be no transitions in the FSM corresponding to some transition in the CM. Restriction 2 forces the transitions to yield a meaning that carries maximal information among the possible meanings. It might be the case that a transition emanating from a source state entails several meanings, none of which entails the rest. In this case, no meaning has the maximal information, and no corresponding transition is included in δ. In order to ensure that there is always a meaning with the most information, we need to ensure that the meaning set $\mathbf{M}$ of the CM is closed under antecedence, that is, for any set of meanings there is a meaning that is stronger than each meaning in the set.

Definition 8. A *complete* CM is a CM whose meaning set $\mathbf{M}$ and final meaning set $\mathbf{F}$ are closed under antecedence. Formally, $(\forall R \subseteq \mathbf{M} : (\exists m_k \in \mathbf{M}: (\forall m_i \in R : m_k \vdash m_i)))$ and $(\forall R \subseteq \mathbf{F}: (\exists m_k \in \mathbf{F}: (\forall m_i \in R : m_k \vdash m_i)))$. ■

A complete CM guarantees that in any subset of both the meaning set $\mathbf{M}$ and the final meaning set $\mathbf{F}$, there exists a meaning that entails all the meanings in the subset.

The main idea underlying a CM execution is that instead of specifying protocols in terms of legal sequences of actions, a CM specifies them in terms of meanings to reach. Thus, two computations may follow different sequences of actions but still achieve the same meaning. Since the computations are characterized with the achieved meanings, rather than pure sequences of actions, computations generated by a CM can be compared semantically.

Definition 9. A computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ is *semantically superior* to a computation $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ (with the same action sequence as τ' if and only if $(\forall i \in \mathbf{J}$: $m'_i \vdash m_i)$. This is written as $\tau' \succeq \tau$. ■

Definition 10. A computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ generated by a CM X is the *semantically strongest* computation if and only if for all computations $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ generated by X (that involve the same action sequence as τ'), τ' is semantically superior to τ. ▮

Definition 11. A CM X' is *semantically superior* to a CM X, written $X' \succeq X$, if and only if ($\forall \tau : \tau$ is generated by $X \Rightarrow (\exists \tau' : \tau'$ is generated by X' and $\tau' \succeq \tau$)). ▮

In the following discussion, we provide two procedures to convert one computation into another. First, Procedure 2 transforms a given computation into the semantically strongest computation based on a particular action sequence.

Procedure 2. Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by a complete CM $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$. We construct the computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ in which $m'_0 = m_0$ and m'_{i+1} is the strongest state that follows m'_i and $\langle a_i : e_i \rangle$. By the definition of a complete CM (Definition 8), we know that such an m'_{i+1} exists. ▮

Lemma 1. Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a complete CM and let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by X. Then Procedure 2 on τ yields a computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ which is the semantically strongest computation (Definition 10) on $\langle a_0, a_1, \ldots, a_{f-1} \rangle$.
Proof. In Procedure 2, after each action a_i at state m'_i, τ' will move to a new state m'_{i+1}, such that m'_{i+1} is the strongest state that can result from doing action a_i in m'_i. Since each m'_i is the strongest state (by the inductive hypothesis), τ'_i is the strongest computation for the given sequence of actions. ■

Recall that in compiling a CM into an FSM, we have not allowed computations to transition from a meaning to itself. Although a compiled FSM does not allow such a transition, a CM does allow it. In other words, a CM may possibly contain redundant transitions. To classify computations that do not have redundant transitions, we introduce the concept of *efficient* computations. A computation is *efficient* if and only if it contains no consecutively repeated states. Procedure 3 transforms a given computation into an efficient computation.

Procedure 3. Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by CM $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$. We produce the computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ that does not have any equivalent consecutive states. That is, we start by copying m_0 to τ'. Iteratively, we check whether applying a_i from state m_i move the computation to a new meaning m_{i+1}. If that is the case, we copy both a_i and m_{i+1} to τ'. Otherwise, we skip a_i and continue the iteration with a_{i+1}. ▮

Lemma 2. Procedure 3 yields an efficient computation.
Proof. Since Procedure 3 removes consecutively repeated states, the resulting computation is efficient. ■

Procedure 3 can transform a computation into one that is shorter. Thus, step-by-step comparisons among computations would not apply. For this reason, we introduce the notion of endpoint equivalence. This notion matches our basic intuition of flexible execution because we only care about where a computation ends, not what intermediate states it went through.

Definition 12. A computation $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ is *endpoint equivalent* to computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_{f'} \rangle$ if and only if $m_0 \equiv m'_0$ and $m_f \equiv m'_{f'}$. Notice that the computations may be of different lengths. ∎

Lemma 3. Procedure 3 preserves endpoint equivalence of computations.
Proof. Procedure 3 produces a computation τ' by removing redundant transitions from a computation τ. Any transition that results in a new meaning is kept. Thus, the last state in τ' equals the last state in τ. Hence, endpoint equivalence is preserved. ∎

Importantly, Lemma 3 shows that Procedure 3 preserves the property of a computation being the semantically strongest for its actions. That is, although the resulting computation has fewer actions, it is the strongest for the actions in it if the input computation had that property in the first place.

Lemma 4. If the computation τ given as input to Procedure 3 is semantically strongest, the computation τ' produced by Procedure 3 is also semantically strongest. ∎

Lemma 5. Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a complete CM compiled into an FSM $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ according to Procedure 1. Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by X, such that τ is efficient semantically strongest, and begins from true. Then τ can be realized by Y.
Proof. By Procedure 1, s_0 = true. Since τ is efficient, we know there are no consecutively repeated states. Thus no transition will be disallowed by Restriction 1. Also, since τ is semantically strongest, each state in τ will be the strongest possible state. Recall that Restriction 2 enforces this requirement on transitions of FSMs. Since Restriction 1 is never exercised, and Restriction 2 does not cause deviation from the flow of τ, τ can be realized by Y. ∎

We pointed out above that with the general definition of CMs, our compilation is not complete. However, complete CMs can easily be found by making the meaning set closed under conjunction. After restricting the class of CMs to complete CMs, we can achieve the following completeness result.

Theorem 3. Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a complete CM compiled into an FSM $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$. Then for any computation that is generated by X and begins from true, there exists an efficient, semantically strongest computation realized by Y.
Proof. Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by X. Let $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ be a computation produced by Procedure 2 when given τ as input. By Lemma 1, we know τ' is semantically strongest. If we give τ' as input to Procedure 3, this yields a computation that is efficient (Lemma 3), and semantically strongest (Lemma 4). Further, this computation can be realized by Y (Lemma 5). ∎

5 Discussion

Commitments have been studied before [3,8], but were not used for protocol specification as we have done here. Commitment machines provide flexibility by capturing the semantic content of the actions in a protocol. By specifying communication protocols using commitments, we can analyze the interactions among participants through the intrinsic meaning of those interactions.

Verharen [15] develops a contract specification language, CoLa, to specify transactions and contracts. Verharen's approach benefits from commitments in expressing actions, but it treats commitments as simple, undirected obligations, and does not allow manipulation of commitments, as in our approach. Further, Verharen only considers base-level commitments, without capturing conditional commitments as we have done.

Dignum and van Linder [5] propose a framework for social agents based on dynamic logic, in which they distinguish messages based on speech acts. In addition to employing commitments, they use *directions* and *declarations* to denote the semantic content of messages. Further, they employ an authority relation between agents to decide on the success of directions and declarations. In our work, we assumed peer-to-peer interactions, in that we do not consider the interactions based on different authority among agents.

d'Inverno *et al.* [6] develop interaction protocols for the multiagent framework, Agentis. They model protocols as a composition of various services and tasks requested and offered among agents. d'Inverno *et al.*'s protocol model consist of four levels: registration, service, task and notification. In all levels of Agentis, the protocols are specified with FSMs, which are formal and simple, but low level.

Smith *et al.* [13] develop protocols in which actions are given a content based on joint intentions. We agree with them on the necessity of declarative content. They model the content of actions with mental attributes whereas we use social constructs. They seem to informally map the joint intentions of the agents to particular states of a finite state machine, but their compilation procedure is not clear. Conversely, our compilation procedure is precise to the level of states, and its soundness and completeness has been established.

Pitt and Mamdani [10] develop an agent communication language (ACL) framework in terms of protocols, and show how an agent replies to a communication by choosing one of the communications allowed by the given communication. They give content to messages based on social constructs, similar to the present approach.

References

1. J. L. Austin. *How to Do Things with Words*. Clarendon Press, Oxford, 1962.
2. M. Barbuceanu and M. S. Fox. COOL: A language for describing coordination in multi agent systems. In *Proc. of the International Conf. on Multiagent Systems*, pages 17–24, 1995.
3. C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proc. of the International Conf. on Multiagent Systems*, pages 41–48, 1995.
4. R. S. Cost *et al.* Using colored Petri nets for conversation modeling. In Frank Dignum and Mark Greaves, ed., *Issues in Agent Communication*, LNAI 1916, pages 178–192, 2000.
5. F. Dignum and B. van Linder. Modeling social agents: Communication as action. In *Intelligent Agents III: Agent Theories, Architectures, and Languages*, pages 205–218, 1997.
6. M. d'Inverno *et al.* Interaction protocols in Agentis. In *Proc. of the 3rd Int. Conf. on Multiagent Systems*, pages 112–119, 1998.
7. M. Fisher and M. Wooldridge. On the formal specification and verification of multi-agent systems. *Int. Journal of Intelligent and Cooperative Information Systems*, 6(1):37–65, 1997.
8. L. Gasser. Social conceptions of knowledge and action: DAI foundations and open systems semantics. In *[9]*, pages 389–404. 1998. (Reprinted from *Artificial Intelligence, 1991*).

9. M. N. Huhns and M. P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, 1998.
10. J. Pitt and A. Mamdani. A protocol-based semantics for an agent communication language. In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, pages 486–491, 1999.
11. M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.
12. M. A. Sirbu. Credits and debits on the Internet. In *[9]*, pages 299–305. 1998. (Reprinted from *IEEE Spectrum, 1997*).
13. I. A. Smith *et al.* Designing conversation policies using joint intention theory. In *Proc. of the 3rd International Conf. on Multiagent Systems*, pages 269–276, 1998.
14. M. Venkatraman and M. P. Singh. Verifying compliance with commitment protocols: Enabling open web-based multiagent systems. *Autonomous Agents and Multi-Agent Sys.*, 2(3):217–236, 1999.
15. E. M. Verharen. *A Language-Action Perspective on the Design of Cooperative Information Agents*. Catholic University, Tilburg, Holland, 1997.

A Semantics

The meanings of formulas generated from L in our BNF grammar are given relative to a model and a state in the model. The meanings of formulas generated from Protocol are given relative to a path and a state on the path. The boolean operators are standard. Useful abbreviations include $\mathsf{false} \equiv (p \wedge \neg p)$, for any $p \in \Phi$, $\mathsf{true} \equiv \neg\mathsf{false}$, $p \vee q \equiv \neg(\neg p \wedge \neg q)$ and $p \rightarrow q \equiv \neg p \vee q$.

$M = \langle \mathbf{S}, <, \approx, \mathbf{N}, \mathbf{A}, \mathbf{C} \rangle$ is a formal model for $\mathcal{P}$. $\mathbf{S}$ is a set of states; $< \subseteq S \times S$ is a partial order indicating branching time, $\approx \subseteq S \times S$ relates states to similar states, and $\mathbf{N} : \mathbf{S} \mapsto 2^{\Phi}$ is an interpretation, which tells us which atomic propositions are true in a given state. $\mathbf{P}$ is the set of paths derived from $<$. $\mathbf{PP}$ gives the powerset of $\mathbf{P}$. For $t \in \mathbf{S}$, $\mathbf{P}_t$ is the set of paths emanating from t. $\mathbf{A}$ is a set of agents. $\mathbf{C} : \mathbf{S} \times \mathbf{A} \times \mathbf{A} \mapsto \mathbf{PP}$ give the modal accessibility relations for commitments, respectively.

For p derived from Protocol, $M \models_t p$ expresses "M satisfies p at t" and for p derived from P, $M \models_{P,t} p$ expresses "M satisfies p at t along path P."

M1. $M \models_t \psi$ iff $\psi \in \mathbf{N}(t)$, where $\psi \in \Phi$
M2. $M \models_t p \wedge q$ iff $M \models_t p$ and $M \models_t q$
M3. $M \models_t \neg p$ iff $M \not\models_t p$
M4. $M \models_t p \rightsquigarrow q$ iff $M \models_t p$ and $(\forall t' : M \models_{t'} p \Rightarrow (\forall t'' : t' \approx t'' \Rightarrow M \models_{t''} q))$
M5. $M \models_t \mathsf{C}(x, y, p)$ iff $(\forall P : P \in \mathbf{C}(x, y, t) \Rightarrow M \models_{P,t} p)$

Generating Bids for Group-Related Actions in the Context of Prior Commitments

Luke Hunsberger

Division of Engineering and Applied Sciences, Harvard University
33 Oxford St., Cambridge, MA 02138, USA
luke@eecs.harvard.edu

Abstract. This paper addresses the problem of bid generation for a group decision-making mechanism based on a combinatorial auction in which agents bid on sets of tasks in a proposed group activity. It presents algorithms that an agent may use to generate temporal constraints for bids in such auctions. These constraints allow an agent making a bid to protect its private schedule of pre-existing commitments in case the bid is ultimately awarded. A task-integration-scheduling algorithm determines whether a new set of tasks can be merged into an agent's private schedule of pre-existing commitments. A temporal-constraint-generation algorithm generates the weakest set of temporal constraints (for inclusion in a bid) sufficient to protect the agent's modified schedule from any possible auction outcome. A proof of correctness for the temporal-constraint-generation algorithm is given. Both algorithms employ Simple Temporal Networks. Although the algorithms are presented in the context of a particular group decision-making mechanism, they have broader applicability to scenarios in which one agent makes an offer to other agents.

Keywords: Collaborative Planning, Group Decision Making, Combinatorial Auctions, Simple Temporal Networks, Bid Generation, Temporal Constraints.

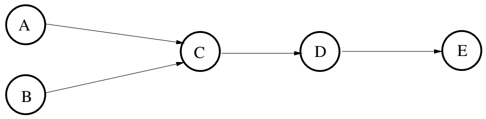
1 Introduction

Suppose a group of agents have been presented with an opportunity to do some group activity α for some reward R. To decide whether they should commit to doing α, they engage a group decision-making mechanism based on a combinatorial auction, as presented by Hunsberger and Grosz [6]. Agents bid on sets of tasks in the proposed group activity. To enable them to protect their private schedules of pre-existing commitments, agents are allowed to include temporal constraints with their bids. When contemplating making a bid, an agent must do the following:

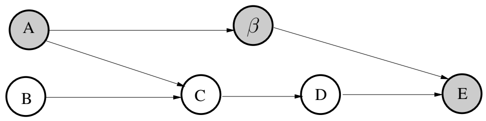
(1) Select a set of tasks to bid on;
(2) Determine ways to integrate those tasks into its private schedule of pre-existing commitments (e.g., by inserting ordering constraints among tasks);
(3) Based on (2), compute the temporal constraints to include with the bid; and
(4) Decide how much it wants to be paid for doing the tasks covered by the bid.

When all the bids are in, a winner-determination algorithm [6,7,3] is run to determine if some combination of non-conflicting bids covers all the tasks in the proposed activity for a total cost that is less than the offered reward. If so, the agents may profit by committing to the task.

Example. A group of agents is presented with an opportunity to build a tool shed this afternoon for a reward of $1000. The tasks include: (A) buying the wood, (B) fetching the tools, (C) sawing the wood, (D) building the shed, and (E) cleaning up, as depicted below with arrows representing ordering constraints.



Suppose Bill is already committed to doing a chair-reupholstering job (β) sometime this afternoon, but nonetheless is considering submitting a bid for the two tasks, A and E. First, Bill must integrate the two tasks A and E into his existing schedule. Suppose that Bill decides it would be best for him to do A first, then β, then E, as shown below, where the shaded tasks are to be done by Bill. (Even though Bill is not bidding on B, C or D, they must appear in his schedule since constraints involving those actions might affect when A and E are to be done.)



Next, Bill must determine what temporal constraints to place in his bid. He can't risk not putting any constraints in his bid because he must protect his pre-existing commitment to doing β. However, he can't include constraints such as "A must be finished before β begins" because the rules of the auction stipulate that constraints in a bid may refer only to the actions being bid on, not to arbitrary actions in the bidder's private schedule. Knowing that β will take at most two hours, Bill makes his bid contingent on E starting at least two hours after A finishes. Finally, Bill must submit an amount he wishes to be paid for doing A and E. Given that doing A and E would substantially impact his schedule, Bill adds $20 on top of the $100 in fixed costs he associates with A and E.

Algorithms Presented in this Paper. In earlier work, Hunsberger and Grosz [6] focused on the Winner Determination part of the auction-based group decision-making mechanism described above. In this paper, we focus on the bid-generation problem. In particular, we focus on how agents may generate appropriate temporal constraints for their bids. We present two algorithms, both of which manipulate Simple Temporal Networks (STNs):

- a task-integration scheduling algorithm that an agent may use to determine whether it can take on a new set of tasks; and
- a temporal-constraint-generation algorithm than an agent may use to generate temporal constraints to include in its bid for that set of tasks.

We assume that the problem of which tasks to bid on can be handled by random selection with backtracking. As for computing the desired compensation for a bid, Horty and Pollack [5] have presented an algorithm an agent might use to compute the cost of taking on new responsibilities in the context of pre-existing commitments. In addition, the measure of the *rigidity* of an agent's schedule, presented in this paper, may be used to generate additional cost information based on the impact on an agent's schedule from taking on new responsibilities.

The Task-Integration-Scheduling Algorithm. The task-integration-scheduling (TIS) algorithm is responsible for determining whether a set of new tasks may be merged into an agent's private schedule of pre-existing commitments. The TIS algorithm is allowed to solve this problem by inserting temporal constraints into the agent's schedule. The TIS algorithm presented in this paper maintains the actions the agent is already committed to in a fully-ordered list. (Alternative TIS algorithms might manipulate partially-ordered, rather than fully-ordered commitments.) New actions are integrated into the agent's schedule by inserting ordering constraints sufficient to ensure that the expanded set of actions will be fully ordered, and hence will not temporally overlap. (We assume an agent can only do one primitive action at a time.) The algorithm is a greedy algorithm based on a heuristic that measures the rigidity of the agent's schedule. The algorithm incrementally adds ordering constraints according to which constraint minimizes the resulting rigidity of the STN representing the agent's schedule.

The Temporal-Constraint-Generation Algorithm. The temporal-constraint-generation (TCG) algorithm presumes that the agent has already integrated the tasks being bid on into its private schedule of pre-existing commitments, but is independent of the particular TIS algorithm being used. The TCG algorithm generates constraints among the tasks being bid on sufficient to ensure that none of the task-integration constraints in the agent's private schedule are violated, no matter how the auction ultimately turns out.

An important aspect of the temporal-constraint-generation problem is that the auctioneer typically combines temporal constraints from multiple bids. From the perspective of an individual agent, it appears that the auctioneer is allowed to strengthen (but not violate) temporal constraints associated with the proposed activity. The TCG algorithm manipulates a pair of STNs, one representing the proposed group activity in the context of the agent's schedule, the other representing the proposed group activity in an empty context. The TCG algorithm compares these STNs to generate the weakest set of temporal constraints sufficient to effectively shield the agent's pre-existing schedule from any strengthening of constraints that the auctioneer might add.

A proof of correctness for the TCG algorithm is given.

2 The Setup

2.1 The Opportunity

Time Points. A *time point (TP)* is a point on the real number line. A *time-point variable* is a variable x that takes on real values.

Temporal Constraints. We consider both unary and binary temporal constraints. A *unary temporal constraint (UTC)* is of the form $L \leq x$ or $x \leq U$, where x is a time-point variable, and L and U are real constants (positive, negative or zero). A *binary temporal constraint (BTC)* is of the form $x_j - x_i \leq \delta$, where x_i and x_j are time-point variables, and the *offset* δ is a constant (positive, negative or zero).

Actions. Following Grosz and Kraus [4], an *act type* represents an abstract class of *act instances* (or *actions*). A *basic action* is a primitive action that a single agent may execute at will (under appropriate conditions), whereas a complex action (whether done by a single agent or a group of agents) is executed by executing the set of subacts in a recipe (described below). We assume that each *act type* τ (whether basic or complex) includes a begin time point b_τ and an end time point e_τ, as well as binary temporal constraints on those time points (e.g., to constrain the duration of the action). Unary temporal constraints are not necessary for general-purpose act types (or recipes).

Recipes. A recipe R_α for a complex act type α consists of a set of actions (called *subacts*) and constraints on those subacts such that the execution of the subacts under those constraints constitutes the doing of α [4]. In this paper, we restrict attention to recipes whose subacts are *basic* actions and whose constraints are purely *temporal*. The temporal constraints specified by the recipe may constrain time points defined by the act type α or the recipe R_α.

Opportunities. An *opportunity* OPP$_\alpha$ specifies a (complex, multi-agent) act type α, a recipe R_α for actions of type α, and sets of unary and binary temporal constraints (beyond those defined in α and R_α). Unlike the general-purpose act types and recipes that do not require unary constraints, an opportunity typically needs them. For example, an opportunity to do a `BUILD-SPACE-STATION` activity might have a unary constraint specifying that the activity be completed before January 1, 2016.

The above definitions are formalized in Figure 1.

2.2 The Combinatorial-Auction-Based Group Decision-Making Mechanism

When a group of agents are confronted with an opportunity to engage in some group activity, they must decide whether they are able to carry out the proposed activity and, if so, whether they should commit to doing so. Hunsberger and Grosz [6] called this the Initial Commitment Decision Problem (ICDP) and presented a group decision-making mechanism based on a combinatorial auction that agents could use to solve the ICDP. Such an auction has a bidding phase and a winner-determination phase [6,7,3]. In the bidding phase, each agent submits zero or more bids, each of which includes a set of subacts being bid on and a set of temporal constraints, as follows.

<u>BID: B</u>

Subacts covered by the bid:	Subacts(B)
Temporal Constraints:	$\mathcal{TC}(B) = \mathcal{UTC}(B) \cup \mathcal{BTC}(B)$,

where $\mathcal{UTC}(B) =$ Unary constraints over $\mathcal{TP}(\alpha) \cup \mathcal{TP}(R_\alpha)$
and $\mathcal{BTC}(B) =$ Binary constraints over $\mathcal{TP}(\alpha) \cup \mathcal{TP}(R_\alpha)$

TEMPORAL CONSTRAINTS

Unary Constraints:	$L \leq x$	or	$x \leq U$
Binary Constraints:	$x_j - x_i \leq \delta$		

ACT TYPE (basic or complex): τ

Time Points:	$\mathcal{TP}(\tau) = \{b_\tau, e_\tau\}$
Temporal Constraints:	$\mathcal{TC}(\tau) =$ Binary constraints over $\mathcal{TP}(\tau)$

RECIPE: R_α (for act type α)

Subacts:	$\{\beta_1, \ldots, \beta_n\}$
Time Points:	$\mathcal{TP}(R_\alpha) = \{b_{\beta_1}, \ldots, b_{\beta_n}; e_{\beta_1}, \ldots, e_{\beta_n}\}$
Temporal Constraints:	$\mathcal{TC}(R_\alpha) =$ Binary constraints over $\mathcal{TP}(\alpha) \cup \mathcal{TP}(R_\alpha)$

OPPORTUNITY: OPP_α

Act Type:	α
Recipe:	R_α
Temporal Constraints:	$\mathcal{UTC}(\mathrm{OPP}_\alpha) =$ Unary constraints over $\mathcal{TP}(\alpha) \cup \mathcal{TP}(R_\alpha)$
	$\mathcal{BTC}(\mathrm{OPP}_\alpha) =$ Binary constraints over $\mathcal{TP}(\alpha) \cup \mathcal{TP}(R_\alpha)$

TIME POINTS and CONSTRAINTS associated with the OPPORTUNITY OPP_α

$$\mathcal{TP}(\mathrm{OPP}_\alpha) = \mathcal{TP}(\alpha) \cup \mathcal{TP}(R_\alpha)$$
$$\mathcal{TC}(\mathrm{OPP}_\alpha) = \mathcal{TC}(\alpha) \cup \mathcal{TC}(R_\alpha) \cup \mathcal{UTC}(\mathrm{OPP}_\alpha) \cup \mathcal{BTC}(\mathrm{OPP}_\alpha)$$

Fig. 1. Constituents of the Definition of an Opportunity to do α using R_α

Bids may also include a *cost* or *reward*; however, in this paper, we focus on feasibility rather than cost-optimization. Thus, we henceforth ignore the costs of bids.

In the winner-determination phase, the auctioneer seeks an *awardable bid-set*—i.e., a set of bids $\mathcal{B}$ such that:

- each subact in R_α is covered by exactly one bid in $\mathcal{B}$; and
- the temporal constraints associated with the opportunity together with the temporal constraints from the bids in $\mathcal{B}$ are simultaneously satisfiable.

If a bid B is awarded to an agent, then that agent must commit to doing the subacts covered by B. Of crucial importance is that that agent must also respect the constraints from *all* the awarded bids. For example, if my bid to wash the dishes sometime between 1:00 and 3:00 is awarded together with your bid to dry the dishes sometime between 12:00 and 2:00, then I must be committed to washing the dishes *before* 2:00 to enable you to finish drying them by 2:00. If my washing the dishes before 2:00 might conflict with some pre-existing commitment of mine (e.g., an ATAL presentation that might take until 2:00), then the constraints in my bid are too weak; they do not protect my pre-existing commitment. One way to protect the pre-existing commitment would be to strengthen the temporal constraint in my bid. (Other options include allowing some form of disjunction or conditionality in the temporal constraints [11,12,8]; however, such added expressiveness carries substantial computational costs.) For example, by changing my dish-washing bid so that it more tightly constrains the dish-washing to occur between 2:00 and 3:00, I ensure that no matter what other bid is chosen to cover the dish-drying activity, my pre-existing commitment to do the ATAL presentation will be protected.

More formally, the temporal constraints $\mathcal{TC}(B)$ for a bid B must ensure that if the bid B is awarded, then no matter what additional constraints $\mathcal{T}$ the auctioneer might require from the other awarded bids:

$$\begin{array}{rll} \text{If:} & \mathcal{TC}(\text{OPP}_\alpha) \cup \mathcal{TC}(B) \cup \mathcal{T} & \text{is satisfiable,} \\ \text{Then:} & \mathcal{TC}(\text{OPP}_\alpha) \cup \mathcal{TC}(B) \cup \mathcal{T} \cup \mathcal{TC}(\text{PS}) & \text{must also be satisfiable,} \end{array}$$

where $\mathcal{TC}(\text{PS})$ refers to temporal constraints from the agent's private schedule (PS) of commitments. (Note that the only difference between the two sets of temporal constraints is that the latter contains constraints from the agent's private schedule.)

3 Simple Temporal Networks

The algorithms in this paper manipulate Simple Temporal Networks [2]. In this Section, we highlight the elements of STNs we'll need in subsequent Sections.

An STN $\mathcal{S}$ is a set of $n+1$ time-point variables $\mathcal{TP}(\mathcal{S}) = \{x_0, \ldots, x_n\}$ together with a set $\mathcal{TC}(\mathcal{S})$ of binary temporal constraints on those variables. The temporal constraints have the form: $x_j - x_i \leq \delta$. Note that lower values of δ correspond to *stronger* constraints. The "variable" x_0 represents an arbitrary, fixed reference point on the time-line. In this paper, we use $x_0 = 0$. Unary constraints on $x_1, \ldots, x_n$ are represented as binary constraints involving the special variable x_0, as follows.

$$\begin{array}{ccccc} L_i \leq x_i & \Longrightarrow & 0 - x_i \leq -L_i & \Longrightarrow & x_0 - x_i \leq -L_i \\ x_i \leq U_i & \Longrightarrow & x_i - 0 \leq U_i & \Longrightarrow & x_i - x_0 \leq U_i \end{array}$$

A *solution* to an STN is a set of variable assignments: $\{x_0 = 0, x_1 = v_1, \ldots, x_n = v_n\}$ satisfying all the constraints in $\mathcal{TC}(\mathcal{S})$. Two STNs $\mathcal{S}_1$ and $\mathcal{S}_2$ (over the same set of time point variables) are said to be *equivalent* if they admit identical solution sets.

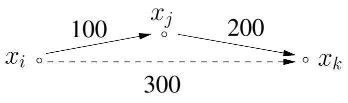
An STN $\mathcal{S}$ may be represented by a weighted, directed graph $G_\mathcal{S} = (V_\mathcal{S}, E_\mathcal{S})$ where the vertices correspond to time points and the edges correspond to binary constraints. More precisely, $V_\mathcal{S} = \mathcal{TP}(\mathcal{S})$ and $E_\mathcal{S} = \{(x_i, \delta, x_j) : (x_j - x_i \leq \delta) \in \mathcal{TC}(\mathcal{S})\}$. In other words, each binary constraint $x_j - x_i \leq \delta$ is represented by a directed edge from x_i to x_j with weight δ. For example, the pair of constraints $x_j - x_i \leq 50$ and $x_i - x_j \leq -12$ are represented by the edges shown below.

$$\begin{tikzcd} x_i \arrow[r, bend left, "50"] & x_j \arrow[l, bend left, "-12"] \end{tikzcd}$$

While the maximum number of explicit constraints in $\mathcal{TC}(\mathcal{S})$ is $n(n+1)$, in practice $\mathcal{TC}(\mathcal{S})$ often contains many fewer constraints. The reason is that even a limited set of *explicit* constraints may give rise to numerous *implicit* constraints. For example, the explicit constraints $x_j - x_i \leq 100$ and $x_k - x_j \leq 200$ combine (using simple arithmetic) to entail the implicit constraint $x_k - x_i \leq 300$, as follows.

$$x_k - x_i \;=\; (x_k - x_j) + (x_j - x_i) \;\leq\; 200 + 100 \;=\; 300$$

In the graphical representation this may be stated as: the "shortest path" from x_i to x_k is no longer than the path from x_i to x_k via x_j:



An STN $\mathcal{S}$ is consistent (i.e., has a solution) if and only if its corresponding graph $G_{\mathcal{S}}$ has no negative cycles [2] (i.e., the path length around any loop is never negative).

The *distance* from x_i to x_j in an STN $\mathcal{S}$ is the *smallest* value δ such that the constraint $x_j - x_i \leq \delta$ is entailed by the constraints in $\mathcal{S}$. (It is appropriate to use the term "distance" for this quantity because it satisfies many of the properties normally associated with a distance function. However, it is important to keep in mind that the distance from x_i to x_j in $\mathcal{S}$ may be negative. In addition, the distance from x_i to x_j is typically not the same as the distance from x_j to x_i.) The distance from x_i to x_j is given by:

$$\text{Distance}(x_i, x_j) \;=\; \mathit{lub}\{(x_j - x_i) \text{ in any solution to } \mathcal{S}\}$$

This quantity is also referred to as the *strongest implicit constraint* from x_i to x_j in $\mathcal{S}$.

The *distance matrix* for an STN $\mathcal{S}$ is an $(n+1)$-by-$(n+1)$ matrix $\mathcal{D}_{\mathcal{S}}$ whose entries record the distance between each pair of time-point variables in $\mathcal{S}$:

$$\mathcal{D}_{\mathcal{S}}(i,j) \;=\; \text{Distance}(x_i, x_j).$$

Since the distance from x_i to x_j in $\mathcal{S}$ corresponds to the length of the shortest path from x_i to x_j in the graph $G_{\mathcal{S}}$, the distance matrix may be computed in polynomial time, for example, using Floyd-Warshall's $O(n^3)$ "all-pairs shortest-path" algorithm [1].

Equivalent STNs have identical distance matrices. In a consistent STN, it is always the case that: $\mathcal{D}_{\mathcal{S}}(i,j) + \mathcal{D}_{\mathcal{S}}(j,i) \;\geq\; 0$. If $\mathcal{D}_{\mathcal{S}}(i,j) + \mathcal{D}_{\mathcal{S}}(j,i) \;=\; 0$, the values x_i and x_j are said to be *rigidly connected* (because their difference is constrained to be a constant). In a consistent STN, it is always possible to add a constraint of the form $x_j - x_i \leq -\mathcal{D}_{\mathcal{S}}(j,i)$ without introducing any inconsistency [2].

4 Solving the Temporal-Constraint-Generation (TCG) Problem

In this section, we define the Temporal-Constraint-Generation Problem and provide an algorithm that solves it. We also provide a proof of correctness for our TCG algorithm.

Inputs:
- An opportunity OPP_α to do an action of type α using a recipe R_α, as in Figure 1;
- Sets of time points $\mathcal{TP}(\mathrm{PS})$ and temporal constraints $\mathcal{TC}(\mathrm{PS})$ representing the agent's private schedule of commitments *prior* to any consideration of OPP_α;
- A set $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ of subacts being bid on (i.e., a subset of the subacts from the recipe R_α); and
- A set $\mathcal{TC}(\Gamma)$ of additional temporal constraints generated by the agent's private task-integration-scheduling (TIS) algorithm (to ensure the schedulability of the subacts in Γ within the agent's pre-existing schedule of commitments).[1]

Goal: Find the weakest set of constraints $\mathcal{TC}(B)$ to include with the bid B (for the subacts in Γ) such that no matter what additional constraints are added as a result of the auction (according to the rules stipulated earlier), the agent's schedule of existing commitments will be protected.

Fig. 2. The Temporal-Constraint-Generation Problem

4.1 The TCG Problem

The Temporal-Constraint-Generation Problem is defined in Figure 2. Note that the constraints in $\mathcal{TC}(B)$ are required to be both necessary and sufficient for protecting the agent's schedule of pre-existing commitments from any potential auction outcome.

4.2 An Algorithm for Solving the TCG Problem

From the information in Figure 2, two STNs are constructed. The first, called $\mathcal{S}_\alpha$, represents the time points and temporal constraints bundled with the opportunity OPP_α. Thus, $\mathcal{S}_\alpha$ is defined by: $\mathcal{TP}(\mathcal{S}_\alpha) = \mathcal{TP}(\mathrm{OPP}_\alpha)$ and $\mathcal{TC}(\mathcal{S}_\alpha) = \mathcal{TC}(\mathrm{OPP}_\alpha)$.

The second, called $\mathcal{S}_{\mathrm{Agt}}$, represents: (1) the time points and temporal constraints associated with the agent's private schedule of pre-existing commitments, (2) the time points and temporal constraints bundled with the opportunity, and (3) the additional temporal constraints inserted to ensure schedulability of the subacts being bid on. Thus, $\mathcal{S}_{\mathrm{Agt}}$ is defined by:

$$\begin{aligned}\mathcal{TP}(\mathcal{S}_{\mathrm{Agt}}) &= \mathcal{TP}(\mathrm{OPP}_\alpha) \cup \mathcal{TP}(\mathrm{PS}) \\ \mathcal{TC}(\mathcal{S}_{\mathrm{Agt}}) &= \mathcal{TC}(\mathrm{OPP}_\alpha) \cup \mathcal{TC}(\mathrm{PS}) \cup \mathcal{TC}(\Gamma).\end{aligned}$$

For convenience, we rename the time-point variables of $\mathcal{S}_\alpha$ and $\mathcal{S}_{\mathrm{Agt}}$, indexing them so that the first $(n+1)$ time points of $\mathcal{S}_{\mathrm{Agt}}$ correspond directly to the time points of $\mathcal{S}_\alpha$:

$$\mathcal{TP}(\mathcal{S}_\alpha) = \{x_0, x_1, \ldots, x_n\} \quad \text{and} \quad \mathcal{TP}(\mathcal{S}_{\mathrm{Agt}}) = \{x_0, x_1, \ldots, x_n; y_1, \ldots, y_k\}$$

[1] One such TIS algorithm is presented in the next Section. A key point is that the TCG algorithm presented in this Section does *not* rely on the particular TIS algorithm used to produce $\mathcal{TC}(\Gamma)$.

We refer to the x_i's as the *opportunity time points*. The y_i's, together with the fixed time point x_0, are referred to as the *non-opportunity time points.* (Note that x_0 is thus both an opportunity and non-opportunity time point.) The y_i's by themselves are referred to as the *strictly* non-opportunity time points.

The temporal constraints in $\mathcal{TC}(\text{OPP}_\alpha)$ may refer only to the opportunity time points. Likewise, the constraints in $\mathcal{TC}(\text{PS})$ may refer only to the non-opportunity time points. However, the constraints in $\mathcal{TC}(\varGamma)$, added to ensure the schedulability of the subacts being bid on, may *link* the opportunity and strictly non-opportunity time points. As a result, the opportunity time points in $\mathcal{S}_{\text{Agt}}$ may be more heavily constrained than their counterparts in $\mathcal{S}_\alpha$. Thus, letting $\mathcal{D}_\alpha$ and $\mathcal{D}_{\text{Agt}}$ be the distance matrices for $\mathcal{S}_\alpha$ and $\mathcal{S}_{\text{Agt}}$, respectively, we have that $\mathcal{D}_{\text{Agt}}(i,j) \leq \mathcal{D}_\alpha(i,j)$ for each $i,j \leq n$.[2]

The $(n+1)$-by-$(n+1)$ sub-matrix of $\mathcal{D}_{\text{Agt}}$ corresponding to the opportunity time points plays an important role in the algorithm. Thus, we define $\mathcal{D}_{\text{Agt}|\alpha}$ to be this submatrix, which corresponds to the first $(n+1)$ rows and columns of $\mathcal{D}_{\text{Agt}}$. It trivially follows that $\mathcal{D}_{\text{Agt}|\alpha}(i,j) \leq \mathcal{D}_\alpha(i,j)$ for each $i,j \leq n$.

The constraints in $\mathcal{TC}(\varGamma)$ are part of $\mathcal{S}_{\text{Agt}}$. These constraints may link opportunity and strictly non-opportunity time points. However, the constraints to be bundled with the bid may only refer to opportunity time points. We get around this problem by creating a new STN, called $\mathcal{S}_{\text{Agt}|\alpha}$, that contains only the opportunity time points but whose *explicit* temporal constraints are derived one-to-one from the strongest *implicit* constraints represented in $\mathcal{D}_{\text{Agt}|\alpha}$, as follows.[3]

$$\begin{aligned}\mathcal{TP}(\mathcal{S}_{\text{Agt}|\alpha}) &= \mathcal{TP}(\text{OPP}_\alpha)\\ \mathcal{TC}(\mathcal{S}_{\text{Agt}|\alpha}) &= \{(x_j - x_i \leq \mathcal{D}_{\text{Agt}|\alpha}(i,j)) : i,j \in \{0,1,\ldots,n\}, i \neq j\}\end{aligned}$$

Note that unlike many STNs that contain comparatively few explicit constraints, $\mathcal{S}_{\text{Agt}|\alpha}$ has an explicit constraint between *every* ordered pair of its time points. Note also that corresponding to each edge among opportunity points in $\mathcal{S}_{\text{Agt}}$ there is an edge in $\mathcal{S}_{\text{Agt}|\alpha}$ that is at least as strong. Finally, note that the previously-defined sub-matrix $\mathcal{D}_{\text{Agt}|\alpha}$ is in fact equal to the distance matrix for the just-defined STN $\mathcal{S}_{\text{Agt}|\alpha}$.

We shall see later that bundling the set $\mathcal{TC}(\mathcal{S}_{\text{Agt}|\alpha})$ of temporal constraints with the bid B would be sufficient to protect the agent's schedule of pre-existing commitments from any potential auction outcome; however, we can pare down the set substantially by (carefully) removing redundancies. Before doing so, we explain why we ignore a minor complicating factor known as *rigid components* [9].

A rigid component exists whenever the difference between two (or more) time points is constrained to be some fixed value. Several researchers [9,14, inter alia] have shown that rigid components are easily dealt with by collapsing each set of rigidly connected time points down to a single point. We employ the same technique but to save space

[2] Note that $\mathcal{D}_\alpha$ is an $(n+1)$-by-$(n+1)$ matrix, whereas $\mathcal{D}_{\text{Agt}}$ is an $(n+k+1)$-by-$(n+k+1)$ matrix.

[3] This construction is related to what Dechter et al. call the *distance graph* of an STN [2]. Here, we are creating a distance graph for a *portion* of an STN.

omit further discussion of it. Henceforth, in this paper, it is assumed that there are no rigid components in the STNs under discussion.

Removing Redundant Edges From $\mathcal{S}_{\text{Agt}|\alpha}$. We employ a notion of *dominance* similar to that used by Tsamardinos et al. [9,10]. A fundamental difference is that their notion of dominance is restricted by their focus on the notion of the *dispatchability* of an STN whereas ours is not.

We say that an explicit constraint $x_k - x_i \leq \delta$ is *dominated* in an STN $\mathcal{S}$ if removing that edge would result in no change to the distance matrix for $\mathcal{S}$. If $\mathcal{D}_{\mathcal{S}}(i, k) < \delta$, then the explicit constraint $x_k - x_i \leq \delta$ is strictly dominated by a stronger implicit constraint. For the case where $\delta = \mathcal{D}_{\mathcal{S}}(i, k)$, we use the following Claim.

Claim. If $\delta = \mathcal{D}_{\mathcal{S}}(i, k)$, then, under the assumption that there are no rigid components in $\mathcal{S}$, the explicit constraint $x_k - x_i \leq \delta$ is dominated if and only if there exists some other point x_j such that: $\delta = \mathcal{D}_{\mathcal{S}}(i, j) + \mathcal{D}_{\mathcal{S}}(j, k)$ (i.e., if and only if there exists a dominating path from x_i to x_k that does not include that *edge* from x_i to x_k).

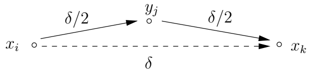
Proof of Claim. If there exists such an x_j, then we can construct a path of length δ by using a shortest path from x_i to x_j followed by a shortest path from x_j to x_k. The absence of rigid components precludes the edge from x_i to x_k being used in either of these shortest paths. Thus, the edge from x_i to x_k does not make a necessary contribution to the STN. On the other hand, if there does not exist such an x_j, then the edge from x_i to x_k represents the only way of constructing a path of length δ from x_i to x_k and hence its removal would alter the distance matrix. ■

The above formula not only provides a convenient way to test for dominance but also tells us that removing a dominated edge will not affect whether some other edge is dominated (because removing a dominated edge does not change the distance matrix and the distance matrix entries are the only things required to determine dominance). Thus, we may use a simple $O(n^3)$ algorithm to locate and remove all dominated edges. (Techniques analogous to those used by Tsamardinos [9] may provide somewhat faster algorithms for tagging dominated edges.)

Let $\mathcal{TC}^u(\mathcal{S}_{\text{Agt}|\alpha})$ be the *undominated* constraints from $\mathcal{S}_{\text{Agt}|\alpha}$.[4] For each (undominated) constraint $x_j - x_i \leq \delta$ in $\mathcal{TC}^u(\mathcal{S}_{\text{Agt}|\alpha})$, if this constraint is *stronger* than the strongest implicit constraint from x_i to x_j in the *opportunity STN* $\mathcal{S}_\alpha$ (i.e., if $\delta < \mathcal{D}_\alpha(i, j)$), then include that constraint with the bid; otherwise, it is not needed. In other words, the set of temporal constraints to bundle with the bid B is given by:

$$\mathcal{TC}(B) = \{(x_j - x_i \leq \delta) \in \mathcal{TC}^u(\mathcal{S}_{\text{Agt}|\alpha}) : \delta < \mathcal{D}_\alpha(i, j)\}.$$

[4] Note that this is *not* the same thing as the undominated constraints among the opportunity time points in $\mathcal{S}_{\text{Agt}}$, as illustrated by the following example which shows an edge (x_i, δ, x_k) that is dominated in $\mathcal{S}_{\text{Agt}}$ but not in $\mathcal{S}_{\text{Agt}|\alpha}$—because the point y_j is a strictly non-opportunity time point and hence does not belong to $\mathcal{S}_{\text{Agt}|\alpha}$:



4.3 Correctness of the TCG Algorithm

Proving the correctness of the TCG algorithm requires showing that the set of constraints $\mathcal{TC}(B)$ is both necessary and sufficient to protect the agent's private schedule of pre-existing commitments from any potential outcome. The sufficiency of the constraints in $\mathcal{TC}(B)$ is given by Theorem 1; the necessity of the constraints in $\mathcal{TC}(B)$ is given by Theorem 2.

Theorem 1. Given the set $\mathcal{TC}(B)$ of temporal constraints generated for a bid B using the TCG algorithm, suppose that the bid B is awarded to the bidder and that the auctioneer inserts an arbitrary set of additional constraints (consistent with the rules of the auction). Then the additional constraints inserted by the auctioneer will be consistent with the agent's schedule of pre-existing commitments (including all constraints concerning the subacts in the bid).

Proof of Theorem 1. Since the theorem assumes that the bid B is awarded to the bidder, we begin by defining $\mathcal{S}_{\alpha^+}$ to be the STN resulting from adding the constraints in $\mathcal{TC}(B)$ to the STN $\mathcal{S}_\alpha$. Let $\mathcal{D}_{\alpha^+}$ be the corresponding distance matrix. Lemmas 1 and 2 below complete the proof. ■

Lemma 1. The distance matrix $\mathcal{D}_{\alpha^+}$ is identical to the distance matrix $\mathcal{D}_{\mathrm{Agt}|\alpha}$. Hence, the corresponding STNs $\mathcal{S}_{\alpha^+}$ and $\mathcal{S}_{\mathrm{Agt}|\alpha}$ are equivalent.

Proof of Lemma 1. Let $x_j - x_i \leq \delta$ be an arbitrary constraint—call it **C**—in $\mathcal{TC}(\mathcal{S}_{\alpha^+}) = \mathcal{TC}(\mathrm{OPP}_\alpha) \cup \mathcal{TC}(B)$. If $\mathbf{C} \in \mathcal{TC}(\mathrm{OPP}_\alpha)$, then $\mathbf{C} \in \mathcal{TC}(\mathcal{S}_{\mathrm{Agt}})$, which implies that $\mathcal{D}_{\mathrm{Agt}}(i,j) \leq \delta$. But $\mathcal{D}_{\mathrm{Agt}}(i,j) = \mathcal{D}_{\mathrm{Agt}|\alpha}(i,j)$, and hence $\mathcal{D}_{\mathrm{Agt}|\alpha}(i,j) \leq \delta$. On the other hand, if $\mathbf{C} \in \mathcal{TC}(B)$, then $\mathbf{C} \in \mathcal{TC}^u(\mathcal{S}_{\mathrm{Agt}|\alpha}) \subseteq \mathcal{TC}(\mathcal{S}_{\mathrm{Agt}|\alpha})$. Thus, in either case, **C** is no stronger than the corresponding (strongest implicit) constraint in $\mathcal{S}_{\mathrm{Agt}|\alpha}$.

In the other direction, let $x_k - x_l \leq \delta'$ be an arbitrary constraint—call it $\mathbf{C}'$—in $\mathcal{TC}(\mathcal{S}_{\mathrm{Agt}|\alpha})$. If $\mathbf{C}' \in \mathcal{TC}(B)$, then by the definition of $\mathcal{S}_{\alpha^+}$, $\mathbf{C}' \in \mathcal{TC}(\mathcal{S}_{\alpha^+})$. Otherwise, according to the definition of $\mathcal{TC}(B)$, either $\delta' \geq \mathcal{D}_\alpha(i,j)$ or $\mathbf{C}' \notin \mathcal{TC}^u(\mathcal{S}_{\mathrm{Agt}|\alpha})$. In the first case, some set of constraints in $\mathcal{TC}(\mathrm{OPP}_\alpha) \subseteq \mathcal{TC}(\mathcal{S}_{\alpha^+})$ entail $\mathbf{C}'$. In the second case, $\mathbf{C}'$ is a dominated constraint in $\mathcal{S}_{\mathrm{Agt}|\alpha}$ which, by definition, means it does not make a necessary contribution to $\mathcal{S}_{\mathrm{Agt}|\alpha}$ and thus may be ignored. In every case, $\mathbf{C}'$ is no stronger than the corresponding (strongest implicit) constraint in $\mathcal{S}_{\alpha^+}$. ■

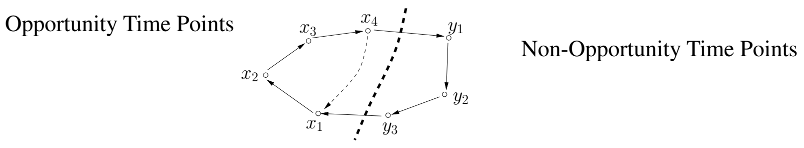
Lemma 2. Let $\mathcal{T}$ be an arbitrary set of temporal constraints over the opportunity time points such that the constraints in $\mathcal{T}$ are *consistent* with the STN $\mathcal{S}_{\alpha^+}$ (i.e., adding that set of constraints to $\mathcal{S}_{\alpha^+}$ would not cause it to become inconsistent). Then that set of constraints is also consistent with $\mathcal{S}_{\mathrm{Agt}}$.

Proof of Lemma 2. Let $\mathcal{S}'_{\alpha^+}$ be the STN resulting from adding the constraints in $\mathcal{T}$ to $\mathcal{S}_{\alpha^+}$. Similarly, let $\mathcal{S}'_{\mathrm{Agt}|\alpha}$ and $\mathcal{S}'_{\mathrm{Agt}}$ be the STNs resulting from adding those same constraints to $\mathcal{S}_{\mathrm{Agt}|\alpha}$ and $\mathcal{S}_{\mathrm{Agt}}$, respectively. By Lemma 1, $\mathcal{S}_{\alpha^+}$ and $\mathcal{S}_{\mathrm{Agt}|\alpha}$ are equivalent; therefore $\mathcal{S}'_{\alpha^+}$ and $\mathcal{S}'_{\mathrm{Agt}|\alpha}$ are equivalent.

Saying that the set of constraints in $\mathcal{T}$ is consistent with $\mathcal{S}_{\alpha+}$ is equivalent to saying that adding the edges corresponding to those constraints would *not* introduce any negative cycles into the graph associated with the STN. In other words, $\mathcal{S}'_{\alpha+}$ does not have any negative cycles (and thus neither does $\mathcal{S}'_{\text{Agt}|\alpha}$, since it is equivalent to $\mathcal{S}'_{\alpha+}$).

Now suppose that, contrary to the statement of the Lemma, the constraints in $\mathcal{T}$ are inconsistent with $\mathcal{S}_{\text{Agt}}$; in other words, suppose that $\mathcal{S}'_{\text{Agt}}$ *does* have a negative cycle.

Case 1. Suppose the negative cycle in $\mathcal{S}'_{\text{Agt}}$ involves at least one strictly non-opportunity time point. We illustrate the method of proof using the following example, leaving it to the reader to carry out the completely straightforward generalization. Consider the cycle shown below (which includes the strictly non-opportunity time points y_1, y_2 and y_3).



Since all of the explicit constraints in $\mathcal{T}$ involve only opportunity time points, all of the edges (i.e., explicit constraints) in $\mathcal{S}'_{\text{Agt}}$ that involve strictly non-opportunity time points are identical to their counterparts in $\mathcal{S}_{\text{Agt}}$. However, the edge from x_4 to x_1 in $\mathcal{S}_{\text{Agt}|\alpha}$ was defined to have the same length as the shortest path from x_4 to x_1 in $\mathcal{S}_{\text{Agt}}$. Thus, if the cycle shown above has negative length in $\mathcal{S}'_{\text{Agt}}$, then the short-circuited cycle (i.e., $x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_1$) must have negative length in $\mathcal{S}'_{\text{Agt}}$—although the "edge" from x_4 to x_1 might only be implicit in $\mathcal{S}'_{\text{Agt}}$. Each edge in the short-circuited cycle is either in $\mathcal{T}$ or in $\mathcal{TC}(\mathcal{S}_{\text{Agt}})$; thus it is either in $\mathcal{T}$ or is dominated by an edge in $\mathcal{TC}(\mathcal{S}_{\text{Agt}|\alpha})$. Thus, that short-circuited cycle must have negative length in $\mathcal{S}'_{\text{Agt}|\alpha}$, which, since the "edge" from x_4 to x_1 is a bona fide edge in $\mathcal{S}'_{\text{Agt}|\alpha}$, contradicts that $\mathcal{S}'_{\text{Agt}|\alpha}$ does not have any negative cycles.

Case 2. Suppose the negative cycle in $\mathcal{S}'_{\text{Agt}}$ involves only opportunity time points. The second half of Case 1 treats exactly this case, leading to a contradiction. ■

Theorem 2. If any of the constraints in $\mathcal{TC}(B)$ are weakened, then the resulting constraint set would be insufficient to protect the agent's private schedule of pre-existing commitments from all potential auction results. In other words, some potential auction result would be inconsistent with the agent's pre-existing commitments.

Prior to proving Theorem 2, we develop some notation and some preliminary results.

Let **C** be an arbitrary constraint $x_j - x_i \leq \delta$ in the set $\mathcal{TC}(B)$.

Result 1. Since $\mathbf{C} \in \mathcal{TC}(B)$, we have that $\delta < \mathcal{D}_\alpha(i,j)$ and hence every path or edge from x_i to x_j in $\mathcal{S}_\alpha$ has length strictly greater than δ. Also, since **C** is undominated in $\mathcal{S}_{\text{Agt}|\alpha}$, we have that $\mathcal{D}_{\text{Agt}|\alpha}(i,j) = \delta$. ■

Let $\mathbf{C}_w$ be the constraint $x_j - x_i \leq \delta_w$, where δ_w is an arbitary value such that $\delta < \delta_w$ (i.e., $\mathbf{C}_w$ is a *weaker* constraint than **C** over x_i and x_j). Let $\mathcal{TC}_w(B)$ be the same as the set of constraints $\mathcal{TC}(B)$ except with **C** replaced by $\mathbf{C}_w$. Similarly, let $\mathcal{S}_{(\mathrm{Agt}|\alpha)_w}$ be the STN that results from replacing **C** by $\mathbf{C}_w$ in $\mathcal{TC}(\mathcal{S}_{\mathrm{Agt}|\alpha})$. (Recall that $\mathbf{C} \in \mathcal{TC}(B) \subseteq \mathcal{TC}(\mathcal{S}_{\mathrm{Agt}|\alpha})$.)

Result 2. By definition, since **C** is undominated in $\mathcal{S}_{\mathrm{Agt}|\alpha}$, removing **C** from $\mathcal{S}_{\mathrm{Agt}|\alpha}$ causes a change to the distance matrix $\mathcal{D}_{\mathrm{Agt}|\alpha}$. However, $\mathcal{S}_{\mathrm{Agt}|\alpha}$ was constructed such that the strongest implicit constraint between each pair of time points was represented by an explicit edge. Therefore, the change to $\mathcal{D}_{\mathrm{Agt}|\alpha}$ caused by removing **C** is limited to the (i, j) entry. Thus, replacing **C** by $\mathbf{C}_w$ yields: $\delta < \mathcal{D}_{(\mathrm{Agt}|\alpha)_w}(i, j) \leq \delta_w$, where $\mathcal{D}_{(\mathrm{Agt}|\alpha)_w}$ is the distance matrix for $\mathcal{S}_{(\mathrm{Agt}|\alpha)_w}$. ■

Let $\mathcal{S}_{\alpha_w^+}$ be the STN that results from adding the constraints in $\mathcal{TC}_w(B)$ to $\mathcal{S}_\alpha$ (i.e., $\mathcal{S}_{\alpha_w^+}$ is the same as $\mathcal{S}_{\alpha^+}$ except that **C** has been replaced by $\mathbf{C}_w$). Let $\mathcal{D}_{\alpha_w^+}$ be the distance matrix for $\mathcal{S}_{\alpha_w^+}$.

Lemma 3. $\mathcal{D}_{\alpha_w^+}(i, j) > \delta$.

Proof of Lemma 3. To the contrary, suppose $\mathcal{D}_{\alpha_w^+}(i, j) \leq \delta$. Then there is some path P from x_i to x_j whose edges are in $\mathcal{TC}(\mathcal{S}_{\alpha_w^+})$ such that: $\mathrm{length}(P) \leq \delta$.

Case 1. Suppose the path P included an *edge* from x_i to x_j. Then P would look like this: $x_i \to \ldots \to x_i \to x_j \to \ldots \to x_j$. Since the length around a loop is always non-negative in a consistent STN (which $\mathcal{S}_{\alpha_w^+}$ is, by assumption), removing the loops from P could only make the length of the path go down or stay the same. Thus, the length of that edge—call it e—from x_i to x_j in $\mathcal{S}_{\alpha_w^+}$ must satisfy: $\mathrm{length}(e) \leq \delta$. However, if $e \in \mathcal{S}_\alpha$ (i.e., if $e \in \mathcal{TC}(\mathrm{OPP}_\alpha)$), Result 1 says that $\mathrm{length}(e) > \delta$. On the other hand, if $e \in \mathcal{TC}_w(B) \subseteq \mathcal{S}_{(\mathrm{Agt}|\alpha)_w}$, Result 2 says that $\mathrm{length}(e) > \delta$. Thus, $e \notin \mathcal{TC}(\mathrm{OPP}_\alpha) \cup \mathcal{TC}_w(B) = \mathcal{TC}(\mathcal{S}_{\alpha_w^+})$, which is a contradiction.

Case 2. Suppose the path P does *not* include an edge from x_i to x_j. Consider an arbitrary edge e on that path. Either $e \in \mathcal{TC}(\mathrm{OPP}_\alpha) \subseteq \mathcal{TC}(\mathcal{S}_{\mathrm{Agt}})$, in which case there is a corresponding edge in $\mathcal{TC}(\mathcal{S}_{Agt|\alpha})$ that is at least as strong as e, or else $e \in \mathcal{TC}_w(B) \subseteq \mathcal{TC}(\mathcal{S}_{\mathrm{Agt}|\alpha})$. In other words, e is either in $\mathcal{TC}(\mathcal{S}_{\mathrm{Agt}|\alpha})$ or is dominated by a corresponding edge in $\mathcal{TC}(\mathcal{S}_{\mathrm{Agt}|\alpha})$. Since e was an arbitrary edge in P, it must be that there is a path in $\mathcal{S}_{Agt|\alpha}$ from x_i to x_j with length less than or equal to δ that does not involve any edge from x_i to x_j, and hence does not involve the edge **C**. But this contradicts that **C** is undominated in $\mathcal{S}_{Agt|\alpha}$. ■

Proof of Theorem 2. Given $\mathcal{S}_{\mathrm{Agt}}$ and $\mathcal{S}_{\alpha_w^+}$ as defined above, we show that there is a constraint $\mathbf{C}^*$ that is consistent with $\mathcal{S}_{\alpha_w^+}$ (and hence is a constraint the auctioneer could legitimately apply), but that is inconsistent with $\mathcal{S}_{\mathrm{Agt}}$ (and hence would be inconsistent with the agent's schedule of pre-existing commitments).

Let $\mathbf{C}^*$ be the constraint $x_i - x_j \leq -\mathcal{D}_{\alpha_w^+}(i, j)$. From Section 3, adding such a constraint is always possible (i.e., it will not affect the consistency of $\mathcal{S}_{\alpha_w^+}$). Thus $\mathbf{C}^*$, which is equivalent to $x_j - x_i \geq \mathcal{D}_{\alpha_w^+}(i, j)$, is a constraint that the auctioneer could legitimately apply. However, by Lemma 3, $\mathcal{D}_{\alpha_w^+}(i, j) > \delta$. Thus, $\mathbf{C}^*$ is inconsistent with the constraint $x_j - x_i \leq \delta$, which is an explicit constraint in $\mathcal{S}_{\mathrm{Agt}|\alpha}$, and hence an implicit constraint in $\mathcal{S}_{\mathrm{Agt}}$. Thus, $\mathbf{C}^*$ is inconsistent with $\mathcal{S}_{\mathrm{Agt}}$. ■

5 The Task-Integration-Scheduling (TIS) Algorithm

In this Section we present a task-integration-scheduling (TIS) algorithm that an agent may use to determine whether a new set of tasks may be merged into its schedule of pre-existing commitments. The algorithm assumes that the actions the agent is already committed to are fully-ordered. The algorithm inserts ordering constraints among the pre-existing commitments and the new tasks being bid on to ensure that the entire collection of tasks forms a fully-ordered list. The algorithm greedily selects the next ordering constraint to add based on a heuristic that measures the resultant change in rigidity in the STN representing the agent's schedule. Recalling that the quantity $\mathcal{D}_\mathcal{S}(i,j) + \mathcal{D}_\mathcal{S}(j,i)$ is always non-negative, but is zero when x_i and x_j are rigidly connected, we define the (average) *rigidity* of an STN by:

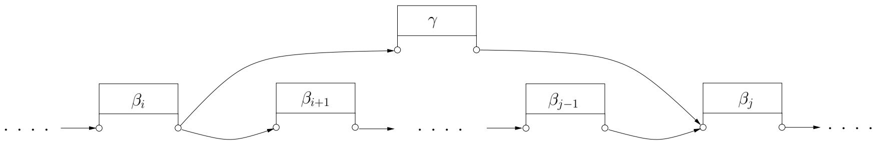
$$\text{Rigidity}(\mathcal{S}) \;=\; \frac{1}{n(n+1)} \sum_{i>j} \frac{1}{1 + \mathcal{D}_\mathcal{S}(i,j) + \mathcal{D}_\mathcal{S}(j,i)}$$

We first show how the algorithm deals with the case of a single new task. We then show the more general case.

TIS Algorithm: Adding a Single Action.

- Scenario: Agent is already committed to doing actions $\beta_1, \ldots, \beta_m$ which are subject to previously-inserted ordering constraints sufficient to ensure that they are fully-ordered (and hence will not overlap). Agent is considering committing to a new action γ which is *not* yet integrated into the fully-ordered list.

1. Agent adds all time points and temporal constraints associated with γ into its STN. If the distance matrix update shows the STN is still consistent, continue; else fail.
2. Agent uses distance matrix to find the maximal i such that β_i is constrained to occur *before* γ, and the minimal j such that β_j is constrained to occur *after* γ.



   If $j = i + 1$, the list is already fully ordered (with γ constrained to occur between β_i and β_{i+1}); exit with success. Otherwise, continue.
3. Agent computes the increase in the STN's rigidity resulting from adding the ordering constraint that γ occur after β_{i+1}, and also from adding the ordering constraint that γ occur before β_{j-1}. If each constraint would cause the STN to become inconsistent, then fail. Otherwise, add the constraint to the STN that results in the minimal increase in rigidity.
4. Go to Step 2.

If the algorithm succeeds, then the agent may commit to doing γ.

TIS Algorithm: Adding Multiple Actions.

- Scenario: Same as above except that agent is considering committing to a *set* of actions: $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$.

1. Agent adds all time points and temporal constraints associated with $\gamma_1, \ldots, \gamma_k$ into its STN. If the distance matrix update shows that the STN is still consistent, continue; else fail.
2. Same as step 2 above, except that it is done for each γ_s in parallel. If some action gets inserted into the fully-ordered list, then remove that action from Γ and redo this step (because there are now more actions in the fully-ordered list from which the i's and j's are chosen). If all actions are fully-ordered (i.e., $\Gamma = \emptyset$), then exit with success; else continue.
3. Same as step 3 above, except that it is done for each γ_s (i.e., there are $2|\Gamma|$ ordering constraints under consideration). If each ordering constraint would cause the STN to become inconsistent, then fail; otherwise choose the ordering constraint that minimizes the increase in the STN's rigidity.
4. Go to Step 2.

We are currently engaged in testing this algorithm in dynamic domains.

6 Conclusions

This paper presents algorithms that an agent may use to generate bids in a combinatorial auction as part of a group decision-making mechanism. The algorithms manipulate Simple Temporal Networks to generate temporal constraints for bids in such auctions. Although the algorithms were designed for this use, they have broader applicability. They may be used to generate protective temporal constraints any time an agent makes an offer to another agent.

The algorithms might be extended to include disjunctive constraints. However, this would be expected to have substantial computational penalties. As an alternative, agents might express limited amounts of disjunction by using multiple bids.

The techniques for manipulating STNs presented in this paper may also be used to describe the flexibility remaining in an agent's schedule while one or more bids are outstanding. This is important if an agent wants to know whether it can adopt additional commitments while waiting for the auction to complete. In addition, the degree to which a bid impacts the flexibility of an agent's schedule may be used to compute part of the bid's contextually-dependent cost.

When the auction mechanism described in this paper succeeds, the agents may still have substantial amounts of temporal reasoning in front of them. For example, if the auction says that Bill must paint the ceiling between 1:00 and 2:00 p.m. and that Bob must wash the brushes after Bill, Bob can't arbitrarily take on outside commitments that force him to wash the brushes at 1:15 p.m. without checking first with Bill. This sort of problem is related to what Vidal [13] calls *contingent* durations (i.e., durations that are determined by external reality, rather than some agent). Generalizing Vidal's notion of *dynamic controllability* to the distributed, multi-agent scenario described in this paper is the subject of current research.

Finally, as the moment of execution approaches, techniques developed by Tsamardinos et al. [9,10] may be integrated to efficiently *dispatch* actions to be executed in real time.

Acknowledgments

The research reported in this paper has been supported in part by NSF grants IRI-9618848, IIS-9978343, IRI 95-25915 and CDA-94-01024.

References

1. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.
2. Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
3. Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 548–553, 1999.
4. Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357, 1996.
5. John F. Horty and Martha E. Pollack. Evaluating new options in the context of existing plans. *Artificial Intelligence*, 127(2):199–220, April 2001.
6. Luke Hunsberger and Barbara J. Grosz. A combinatorial auction for collaborative planning. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 151–158. IEEE Computer Society, 2000.
7. Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.
8. Eddie Schwalb and Rina Dechter. Processing disjunctions in temporal constraint networks. *Artificial Intelligence*, 93:29–61, 1997.
9. Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. Master's thesis, University of Pittsburgh, 2000.
10. Ioannis Tsamardinos, Nicola Muscettola, and Paul Morris. Fast transformation of temporal plans for efficient execution. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. AAAI Press/MIT Press, 1998.
11. Ioannis Tsamardinos, Martha E. Pollack, and John F. Horty. Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches. In *Proceedings of the Artificial Intelligence Planning and Scheduling 2000 Conference (AIPS 2000)*, 2000.
12. Ioannis Tsamardinos, Martha E. Pollack, and Colleen McCarthy. Efficient solution techniques for the disjunctive temporal problem. Technical Report ISP001-01, Univeristy of Pittsburgh, 2001.
13. Thierry Vidal and Hélène Fargier. Contingent durations in temporal CSPs: from consistency to controllabilities. *Linkoping Electronic Articles in Computer and Information Science, ISSN 1401-9841*, 2(2), 1997. Available from: http://www.ep.liu.se/ea/cis/1997/002/.
14. Rattana Wetprasit and Abdul Sattar. Qualitative and quantitative temporal reasoning with points and durations. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. AAAI Press/MIT Press, 1998.

Dynamic Distributed Resource Allocation: A Distributed Constraint Satisfaction Approach

Pragnesh Jay Modi, Hyuckchul Jung, Milind Tambe,
Wei-Min Shen, and Shriniwas Kulkarni

University of Southern California/Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, USA
{modi,jungh,tambe,shen,kulkarni}@isi.edu

Abstract. In distributed resource allocation a set of agents must assign their resources to a set of tasks. This problem arises in many real-world domains such as distributed sensor networks, disaster rescue, hospital scheduling and others. Despite the variety of approaches proposed for distributed resource allocation, a systematic formalization of the problem, explaining the different sources of difficulties, and a formal explanation of the strengths and limitations of key approaches is missing. We take a step towards this goal by proposing a formalization of distributed resource allocation that represents both dynamic and distributed aspects of the problem. We define four categories of difficulties of the problem. To address this formalized problem, the paper defines the notion of Dynamic Distributed Constraint Satisfaction Problem (DyDCSP). The central contribution of the paper is a generalized mapping from distributed resource allocation to DyDCSP. This mapping is proven to correctly perform resource allocation problems of specific difficulty. This theoretical result is verified in practice by an implementation on a real-world distributed sensor network.

1 Introduction

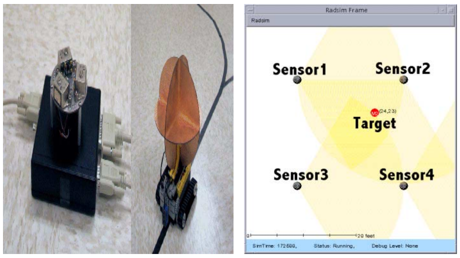
Distributed resource allocation is a general problem in which a set of agents must intelligently assign their resources to a set of tasks such that all tasks are performed with respect to certain criteria. This problem arises in many real-world domains such as distributed sensor networks [7], disaster rescue[4], hospital scheduling[2], and others. However, despite the variety of approaches proposed for distributed resource allocation, a systematic formalization of the problem, explaining the different sources of difficulties, and a formal explanation of the strengths and limitations of key approaches is missing.

We propose a formalization of distributed resource allocation that is expressive enough to represent both dynamic and distributed aspects of the problem. These two aspects present some key difficulties. First, a distributed situation results in agents obtaining only local information, but facing global *ambiguity* — an agent may know the results of its local operations but it may not know the global task and hence may not know what operations others should perform. Second, the situation is dynamic so a solution to the resource allocation problem at one time may become unsuccessful when the underlying tasks have changed. Therefore, the agents must continuously monitor the quality of the solution and must have a way to express such changes in the problem. Given these parameters of ambiguity and dynamism, we will define four classes of difficulties of the problem. In order to address the resource allocation problem, the paper also defines the notion of Dynamic Distributed Constraint Satisfaction Problem (DyDCSP).

The central contribution of the paper is a reusable, generalized mapping from distributed resource allocation to DyDCSP. This mapping is proven to correctly perform resource allocation problems of specific difficulty. This theoretical result is verified in practice by an implementation on a real-world distributed sensor network. Ideally, our formalization may enable researchers to understand the difficulty of their resource allocation problem, choose a suitable mapping using DyDCSP, with automatic guarantees for correctness of the solution.

2 Domains and Motivations

Among the domains that motivate this work, the first is a distributed sensor network. This domain consists of multiple stationary sensors, each controlled by an independent agent, and targets moving through their sensing range (Figure 1.a and Figure 1.b illustrates the real hardware and simulator screen, respectively). Each sensor is equipped with three Doppler radar heads, each covering a 120 degree sector. An agent may activate at most one radar head, or sector, of the sensor it controls at a given time. While all of the agents must act as a team to cooperatively track the targets, there are some key difficulties in such tracking.



(a) sensor(left) and target(right) (b) simulator (top-down view)

Fig. 1. A distributed sensor domain

First, in order for a target to be tracked accurately, at least three agents must concurrently activate overlapping sectors so that the target position can be triangulated. For example, in Figure 2 which corresponds to Figure 1.b, if agent A1 detects target 1 in its sector 0, it must coordinate with neighboring agents, A2 and A4 say, so that they activate their respective sectors that overlap with A1's sector 0. We call activating a sector an "operation". Since there are three sectors of 120 degrees, each agent has three operations. An operation succeeds if a target is detected in that sector and fails otherwise.

Second, there is local ambiguity for an agent when choosing a target to track, so an agent cannot necessarily tell other agents which sectors to activate. This is because each agent can detect only the distance and speed of a target. For example in Figure 2, if agent

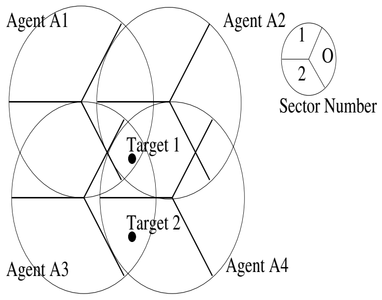


Fig. 2. Each sensor (agent) has three sectors.

A3 activates its sector 0, it will get an identical signal regardless of whether target 1 or 2 is present. Therefore, A3 cannot tell A4 which of its two sectors (sector 1 or sector 2) to activate since it only knows that there is a target somewhere in its own sector 0. That is, a single agent does not know which target, or task, is to be performed. For A3 to identify which task to perform, it must know the result of other related agents' operations.

Third, if there are multiple targets, an agent may be needed to activate more than one sector at the same time, which is not possible. For instance in Figure 2, A4 would need to choose between activating its sector 1 to track target 1 or its sector 2 to track target 2. Thus, the relationship among tasks to perform will affect the difficulty of the resource allocation problem.

Fourth, the situation is dynamic as targets move through the sensing range. The dynamic property of the domain makes problems even harder. After agents activate overlapping sectors and begin tracking a target, as the target moves over time, they may have to find different overlapping sectors.

The second domain which motivates our work is the Robocup Rescue [4] simulator for disaster rescue after an earthquake. Here, multiple fire engines, ambulances and police cars must collaborate to save civilians from trapped, burning buildings and no centralized control is available to allocate all of the resources. For instance, an ambulance must collaborate with a fire engine have a fire extinguished before it can rescue a civilian. The tasks are dynamic, e.g., fires grow or shrink and also ambiguous e.g., a fire engine could receive a report of a fire in an area, but not a specific building on fire. This domain thus presents another example of a distributed resource allocation problem with many similarities with the distributed sensor network problem.

The above applications illustrates the difficulty of resource allocation among distributed agents in dynamic environment. Lack of a formalism for dynamic distributed resource allocation problem can lead to ad-hoc methods which cannot be easily reused.

3 Formalization

A Distributed Resource Allocation Problem consists of 1) a set of agents that can each perform some set of operations, and 2) a set of tasks to be completed. In order to be completed, a task requires some subset of agents (resources) to perform the necessary operations. Thus, we can define a task by the operations that agents must perform in order to complete it. The problem to be solved is an allocation of agents to tasks such that all tasks are performed. More formally, a Distributed Resource Allocation Problem is a structure $\langle \mathcal{A}g, \mathcal{O}, \mathcal{T} \rangle$ where

- $\mathcal{A}g$ is a set of agents, $\mathcal{A}g = \{A_1, A_2, ..., A_n\}$.
- $\mathcal{O} = \{O_1^1, O_2^1, ..., O_p^i, ..., O_q^n\}$ is a set of operations, where operation O_p^i denotes the p'th operation of agent A_i. An operation can either succeed or fail. Let $Op(A_i)$ denote the set of operations of A_i. Operations in $Op(A_i)$ are *mutually exclusive*; an agent can only perform one operation at a time.
- $\mathcal{T}$ is a set of tasks, where a task is a collection of sets of operations ($\mathcal{T} \subseteq$ power set of $\mathcal{O}$). Let T be a task in $\mathcal{T}$. $t_r \in T$ is a set of operations called a *minimal set* because it represents the minimal resources necessary to complete the task. If there are alternative minimal sets that can be used to complete T, we require that the following *minimality* property hold:

$$\forall t_r, t_s \in T, t_r \nsubseteq t_s \; and \; t_s \nsubseteq t_r$$

  Finally, minimal sets from two different tasks *conflict* if they contain operations belonging to the same agent.

A *solution* to a resource allocation problem involves choosing minimal sets for tasks such that the chosen minimal sets do not conflict. To illustrate this formalism in the distributed sensor network domain, we cast each sensor as an agent and activating one of its (three) sectors as an operation. Following our notation, O_p^i denotes the operation where agent A_i activates sector p. For example in Figure 2, we have four agents, so $\mathcal{A}g$ = $\{A_1, A_2, A_3, A_4\}$. Each agent A_i can perform one of three operations, so $Op(A_i)$ = $\{ O_0^i, O_1^i, O_2^i \}$ and $\mathcal{O} = \bigcup_{A_i \in \mathcal{A}g} Op(A_i)$.

Now we have left to define our task set $\mathcal{T}$. We will define a separate task for each target in a particular location, where a location corresponds to an area of overlap of sectors. In the situation illustrated in Figure 2, we have two targets shown, so we define two tasks: $\mathcal{T} = \{T_1, T_2\}$. Since a target requires three agents to track it accurately, task T_1 requires any three of the four possible agents to activate their correct sector, so we define a minimal set for each of the $\binom{4}{3}$ combinations. Thus, $T_1 = \{\{O_0^1, O_2^2, O_0^3\}, \{O_2^2, O_0^3, O_1^4\}, \{O_0^1, O_0^3, O_1^4\}, \{O_0^1, O_2^2, O_1^4\}\}$. Task T_2 can only be tracked by two agents, both of which are needed, so $T_2 = \{\{O_0^3, O_2^4\}\}$ (In reality, more agents are needed).

For each task T_r, we use $\Upsilon(T_r)$ to denote the union of all the minimal sets of T_r, and for each operation, we use $T(O_p^i)$ to denote the set of tasks that include O_p^i in some minimal set. For instance in our example, $\Upsilon(T_1) = \{O_0^1, O_2^2, O_0^3, O_1^4\}$ and $T(O_0^3) = \{T_1, T_2\}$. We will also require that every operation should serve some task, i.e., $\forall\, O_p^i \in \mathcal{O}$, $|\, T(O_p^i) \,| \neq 0$. Formal definitions for Υ and T are as follows:

- $\forall\, T_r \in \mathcal{T}, \Upsilon(T_r) = \bigcup_{t_r \in T_r} t_r$
- $\forall\, O_p^i \in \mathcal{O}, T(O_p^i) = \{T_r \mid O_p^i \in \Upsilon(T_r)\}$

All the tasks in $\mathcal{T}$ may not always be present. We use $\mathcal{T}_{current}$ ($\subseteq \mathcal{T}$) to denote the set of tasks that are currently present. This set is determined by the environment. We call a resource allocation problem *static* if $\mathcal{T}_{current}$ is constant over time and *dynamic* otherwise. So in our distributed sensor network example, a moving target represents a dynamic problem. Agents can execute their operations at any time, but the success of an operation is determined by the set of tasks that are currently present. The following two definitions formalize this interface with the environment.

Definition 1: $\forall\, O_p^i \in \mathcal{O}$, if O_p^i is executed and $\exists\, T_r \in \mathcal{T}_{current}$ such that $O_p^i \in \Upsilon(T_r)$, then O_p^i is said to *succeed.*

So in our example, if agent A_1 executes operation O_0^1 and if $T_1 \in \mathcal{T}_{current}$, then O_0^1 will succeed, otherwise it will fail. Next, a task is performed when all the operations in some minimal set succeed. More formally,

Definition 2: $\forall T_r \in \mathcal{T}$, T_r is *performed* iff $\exists t_r \in T_r$ such that all the operations in t_r succeed. All tasks that satisfy this definition are contained in $\mathcal{T}_{current}$.

Agents must somehow be informed of the set of current tasks. The notification procedure is outside of this formalism. Thus, the following assumption states that at least one agent is notified that a task is present by the success of one of its operations. (This assumption can be satisfied in the distributed sensor domain by agents "scanning" for targets by rotating sectors when they are currently not tracking a target.)

Notification assumption: $\forall T_r \in \mathcal{T}$, if $T_r \in \mathcal{T}_{current}$, then $\exists\, O_p^i \in \Upsilon(T_r)$ such that O_p^i succeeds and $\forall T_s (\neq T_r) \in \mathcal{T}_{current}$, $O_p^i \notin T_s$.

We now define some properties of a given resource allocation problem. Multiple tasks in $\mathcal{T}_{current}$ can be performed concurrently with some conditions. For instance, two different tasks T_r and T_s can be performed concurrently if all the operations for performing T_r can be executed concurrently with those for performing T_s. We define two types of *conflict-free* to denote tasks that can be performed concurrently. The **strongly conflict free** condition implies that all choices of minimal sets from the tasks are non-conflicting. The **weakly conflict free** condition implies that there exists a choice of minimal sets from the tasks that are non-conflicting or in other words, there exists some solution.

Definition 3: A resource allocation problem is called **strongly conflict free (SCF)** if $\forall\, T_r, T_s \in \mathcal{T}_{current}$, the following statement is true:

– if $T_r \neq T_s$, then $\forall\, t_r \in T_r, \forall\, t_s \in T_s, \forall\, A_i \in \mathcal{A}g$, $\mid t_r \cap Op(A_i) \mid + \mid t_s \cap Op(A_i) \mid \leq 1$.

Definition 4: A resource allocation problem is called **weakly conflict free (WCF)** if $\forall\, T_r, T_s \in \mathcal{T}_{current}$, the following statement is true:

– if $T_r \neq T_s$, then $\exists\, t_r \in T_r, \exists\, t_s \in T_s$ s.t. $\forall\, A_i \in \mathcal{A}g$, $\mid t_r \cap Op(A_i) \mid + \mid t_s \cap Op(A_i) \mid \leq 1$.

When there is no centralized control, no agent knows what are the current tasks. When an operation O_p^i of an agent A_i succeeds, the agent only knows that there is an unidentified task to be performed from a task set $\mathcal{T}$. If $\mid T(O_p^i) \mid = 1$, there is no ambiguity in identifying a task to be performed. A_i whose operation succeeds can look up the task and inform other related agents of what the current task is. However, if there are multiple tasks for which O_p^i is required, (if $\mid T(O_p^i) \mid > 1$), A_i has to decide what the task is by communicating with other agents. A task $T_r \in T(O_p^i)$ can be identified when all the operations in a minimal set $t_r \in T_r$ succeed. Since operations from different agents are involved, the decision problem cannot be solved by an individual agent.

The Dynamic Distributed Resource Allocation Problem is to identify current tasks that can change over time and assign operations that are required by the current tasks. The difficulty of the problem depends on both the ambiguity of the tasks from a dynamic distributed environment and the relation among tasks which may require conflicting resources. Here, we outline four problem classes of increasing difficulty based on ambiguity and the relation among tasks.

- **Class 1 (static, strongly conflict free, no ambiguity)**: In this class of problems, the tasks in $\mathcal{T}_{current}$ are fixed, but unknown, and are *strongly conflict free*. Furthermore, $\forall O_p^i \in \mathcal{O}$, $\mid T(O_p^i) \mid= 1$, so there is no ambiguity in identifying a task when an operation succeeds.
- **Class 2 (static, strongly conflict free, ambiguity)**: In this class of problems, the tasks in $\mathcal{T}_{current}$ are fixed, but unknown, and are *strongly conflict free*. Furthermore, $\forall O_p^i \in \mathcal{O}$, $\mid T(O_p^i) \mid\geq 1$, so there may be ambiguity in identifying a task when an operation succeeds.
- **Class 3 (dynamic, strongly conflict free, ambiguity)**: Dynamism means that $\mathcal{T}_{current}$ changes over time. In other words, the set of tasks that must be performed by the agents is dynamic.
- **Class 4 (dynamic, weakly conflict free, ambiguity)**: By relaxing the *strongly conflict free* assumption, we make agents' decision even harder. With *weakly conflict free* tasks, agents may need to select an operation among mutually exclusive operations. Agents must negotiate to find a solution without involving mutually exclusive operations for each task in $\mathcal{T}_{current}$.

As the number of class increases, the problem difficulty also increases and thus any solution method for a class can solve any problem in lower classes.

4 Dynamic DCSP

A Constraint Satisfaction Problem (CSP) is commonly defined by a set of variables, each associated with a finite domain, and a set of constraints on the values of the variables. A solution is the value assignment for the variables which satisfies all the constraints. A distributed CSP is a CSP in which variables and constraints are distributed among multiple agents. Each variable belongs to an agent. A constraint defined only on the variable belonging to a single agent is called a *local constraint*. In contrast, an *external constraint* involves variables of different agents. Solving a DCSP requires that agents not only solve their local constraints, but also communicate with other agents to satisfy external constraints.

DCSP assumes that the set of constraints are fixed in advance. This assumption is problematic when we attempt to apply DCSP to domains where features of the environment are not known in advance and must be sensed at run-time. For example, in distributed sensor networks, agents do not know where the targets will appear. This makes it difficult to specify the DCSP constraints in advance. Rather, we desire agents to sense the environment and then activate or deactivate constraints depending on the result of the sensing action. We formalize this idea next.

We take the definition of DCSP one step further by defining Dynamic DCSP (DyDCSP). DyDCSP allows constraints to be conditional on some predicate P. More specifically, a *dynamic* constraint is given by a tuple (P, C), where P is an arbitrary predicate that is continuously evaluated by an agent and C is a familiar constraint in DCSP. When P is true, C must be satisfied in any DCSP solution. When P is false, C may be violated. An important consequence of dynamic DCSP is that agents no longer terminate when they reach a stable state. They must continue to monitor P, waiting to see if it changes. If its value changes, they may be required to search for a new solution. Note that a solution when P is true is also a solution when P is false, so the deletion of a constraint does not require any extra computation. However, the converse does not hold. When a constraint is added to the problem, agents may be forced to compute a new solution. In this work, we only need to address a restricted form of DyDCSP i.e., it is only necessary that *local constraints* be dynamic.

AWC [8] is a sound and complete algorithm for solving DCSPs and it is described briefly here. An agent with local variable A_i, chooses a value v_i for A_i and sends this value to agents with whom it has external constraints. It then waits for and responds to messages. When the agent receives a variable value ($A_j = v_j$) from another agent, this value is stored in an AgentView. Therefore, an AgentView is a set of pairs $\{(A_j, v_j)$, (A_k, v_k), ...$\}$. Intuitively, the AgentView stores the current value of non-local variables. A subset of an AgentView is a NoGood if an agent cannot find a value for its local variable that satisfies all constraints. For example, an agent with variable A_i may find that the set $\{(A_j, v_j)$, $(A_k, v_k)\}$ is a NoGood because, given these values for A_j and A_k, it cannot find a value for A_i that satisfies all of its constraints. This means that these value assignments cannot be part of any solution. In this case, the agent will request that the others change their variable value and a search for a solution continues. To guarantee completeness, a discovered NoGood is stored so that that assignment is not considered in the future.

The most straightforward way to attempt to deal with dynamism in DCSP is to consider AWC as a subroutine that is invoked anew everytime a constraint is added. Unfortunately, in many domains such as ours, where the problem is dynamic but does not change drastically, starting from scratch may be prohibitively inefficient. Another option, and the one that we adopt, is for agents to continue their computation even as local constraints change asynchronously. The potential problem with this approach is that when constraints are removed, a stored NoGood may now become part of a solution. We solve this problem by requiring agents to store their own variable values as part of non-empty NoGoods. For example, if an agent with variable A_i finds that a value v_i does not satisfy all constraints given the AgentView $\{(A_j, v_j)$, $(A_k, v_k)\}$, it will store the set $\{(A_i, v_i)$, (A_j, v_j), $(A_k, v_k)\}$ as a NoGood. With this modification to AWC, NoGoods remain "no good" even as local constraints change. Let us call this modified algorithm Locally-Dynamic AWC (LD-AWC) and the modified NoGoods "LD-NoGoods" in order to distinguish them from the original AWC NoGoods.

Lemma I: LD-AWC is sound and complete.

The soundness of LD-AWC follows from the soundness of AWC. The completeness of AWC is guaranteed by the recording of NoGoods. A NoGood logically represents a set of assignments that leads to a contradiction. We need to show that this invariant is maintained in LD-NoGoods. An LD-NoGood is a superset of some non-empty AWC NoGood and since every superset of an AWC NoGood is no good, the invariant is true when a LD-NoGood is first recorded. The only problem that remains is the possibility that an LD-NoGood may later become good due to the dynamism of local constraints. A LD-NoGood contains a specific value of the local variable that is no good but never contains a local variable exclusively. Therefore, it logically holds information about external constraints only. Since external constraints are not allowed to be dynamic in LD-AWC, LD-NoGoods remain valid even in the face of dynamic local constraints. Thus the completeness of LD-AWC is guaranteed.

5 Generalized Mapping

In this section, we map the Class 3 Resource Allocation Problem, which subsumes Class 1 and 2, onto DyDCSP. Our goal is to provide a general mapping so that any such resource allocation problem can be solved in a distributed manner by a set of agents by applying this mapping.

Our mapping of the Resource Allocation Problem is motivated by the following idea. The goal in DCSP is for agents to choose values for their variable so all constraints are satisfied. Similarly, the goal in resource allocation is for the agents to choose operations so all tasks are performed. Therefore, in our first attempt we map variables to agents and values of variables to operations of agents. So for example, if an agent A_i has three operations it can perform, $\{O_1^i, O_2^i, O_3^i\}$, then the variable corresponding to this agent will have three values in its domain. However, this simple mapping attempt fails because an operation of an agent may not always succeed. Therefore, in our second attempt, we define two values for every operation, one for success and the other for failure. In our example, this would result in six values.

It turns out that even this mapping is inadequate for the Class 2 and 3 Resource Allocation Problem. This is because an operation can be required for more than one task. We desire agents to be able to not only choose which operation to perform, but also to choose for which task they will perform the operation. For example in Figure 2, Agent A3 is required to activate the same sector for both targets 1 and 2. We want A3 to be able to distinguish between the two targets, so that it does not unnecessarily require A2 to activate sector 2 when target 2 is present. So, for each of the values defined so far, we will define new values corresponding to each task that an operation may serve.

More formally, given a Class 3 Resource Allocation Problem $\langle \mathcal{A}g, \mathcal{O}, \mathcal{T} \rangle$, the corresponding DCSP is defined over a set of n variables,

– $A = \{A_1, A_2, ..., A_n\}$, one variable for each $A_i \in$ Ag. We will use the notation A_i to interchangeably refer to an agent or its variable.

The domain of each variable is given by:

– $\forall A_i \in \mathcal{A}g$, Dom$(A_i) = \bigcup\limits_{O_p^i \in \mathcal{O}} O_p^ixT(O_p^i)$x{yes,no}.

In this way, we have a value for every combination of operation an agent can perform, task for which the operation is required, and whether the operation succeeds or fails.

For example in Figure 2, Agent A3 would have 4 values in its domain for its sector 0: $\{O_0^3T_1yes, O_0^3T_1no, O_0^3T_2yes, O_0^3T_2no\}$.

A word about notation: $\forall\ O_p^i \in \mathcal{O}$, the set of values in O_p^ix$T(O_p^i)$x{yes} will be abbreviated by the term O_p^i*yes and the assignment $A_i = O_p^i$*yes denotes that $\exists v \in O_p^i$*yes s.t. $A_i = v$. Intuitively, the notation is used when an agent detects that an operation is succeeding, but it is not known which task is being performed. This analagous to the situation in the distributed sensor network domain where an agent may detect a target in a sector, but not know its exact location. Finally, when a variable A_i is assigned a value, we assume the corresponding agent is required to execute the corresponding operation.

Next, we must constrain agents to assign "yes" values to variables only when an operation has succeeded. However, in Class 3 problems, an operation may succeed at some time and fail at another time since tasks are dynamically added and removed from the current set of tasks to be performed. Thus, every variable is constrained by the following dynamic local constraints.

- **Dynamic Local Constraint 1 (LC1)**: $\forall T_r \in \mathcal{T}, \forall O_p^i \in \Upsilon T_r)$, we have
  LC1(A_i) = (P, C), where P: O_p^i succeeds.
  C: $A_i = O_p^i$*yes
- **Dynamic Local Constraint 2 (LC2)**: $\forall T_r \in \mathcal{T}, \forall O_p^i \in \Upsilon(T_r)$, we have
  LC2(A_i) = (P, C), where P: O_p^i does not succeed.
  C: $A_i \neq O_p^i$*yes

The truth value of P is not known in advance. Agents must execute their operations, and based on the result, locally determine if C needs to be satisfied. In the Class 1 and 2 problems, the set of current tasks does not change and thus, the truth value of P, although initially unknown, once known will not change over time. On the other hand, in the Class 3 and 4 problems, where the set of current tasks is changing over time, the truth value of P will change, and hence the corresponding DCSP will be truly dynamic.

We now define the external constraint (EC) between variables of two different agents. EC is a normal static constraint and is always present. It is expressed here as a logical implication.

- **External Constraint**: $\forall T_r \in \mathcal{T}, \forall O_p^i \in \Upsilon(T_r), \forall A_j \in A$,

  EC(A_i, A_j): (1) $A_i = O_p^iT_r$yes, and
  (2) $\forall t_r \in T_r$ s.t. $O_p^i \in t_r$, $\exists q$ s.t. $O_q^j \in t_r$.
  $\Rightarrow A_j = O_q^jT_r$yes

The EC constraint requires some explanation. Condition (1) states that an agent A_i has found an operation that succeeds for task T_r. Condition (2) quantifies the other agents whose operations are also required for T_r. If A_j is one of those agents, the consequent requires it to choose its respective operation for T_r. If A_j is not required for T_r, condition (2) is false and EC is trivially satisfied. Finally, note that every pair of variables A_i and A_j, have two EC constraints between them: one from A_i to A_j and another from A_j to A_i. The conjunction of the two unidirectional constraints can be considered one bidirectional constraint.

We will now prove that our mapping can be used to solve any given Class 3 Resource Allocation Problem. The first theorem shows that our DyDCSP always has a solution, and the second theorem shows that if agents reach a solution, all current tasks are performed. It is interesting to note that the converse of the second theorem does not hold, i.e., it is possible for agents to be performing all tasks *before* a solution state is reached. This is due to the fact that when all current tasks are being performed, agents whose operations are not necessary for the current tasks could still be violating constraints.

Theorem I : Given a Class 3 Resource Allocation Problem $\langle \mathcal{A}g, \mathcal{O}, \mathcal{T} \rangle$, $\mathcal{T}_{current} \subseteq \mathcal{T}$, there exists a solution to the corresponding DyDCSP.

proof: We proceed by presenting a variable assignment and showing that it is a solution.

Let $B = \{A_i \in A \mid \exists T_r \in \mathcal{T}_{current}, \exists O_p^i \in \Upsilon(T_r)\}$. We will first assign values to variables in B, then assign values to variables that are not in B. If $A_i \in B$, then $\exists T_r \in \mathcal{T}_{current}, \exists O_p^i \in \Upsilon(T_r)$. In our solution, we assign $A_i = O_p^i T_r yes$. If $A_i \notin B$, we may choose any $O_p^i T_r no \in$ Dom(A_i) and assign $A_i = O_p^i T_r no$.

To show that this assignment is a solution, we first show that it satisfies the EC constraint. We arbitrarily choose two variables, A_i and A_j, and show that EC(A_i, A_j) is satisfied. We proceed by cases. Let $A_i, A_j \in A$ be given.

- *case 1: $A_i \notin B$*
  Since $A_i = O_p^i T_r no$, condition (1) of EC constraint is false and thus EC is trivially satisfied.
- *case 2: $A_i \in B, A_j \notin B$*
  $A_i = O_p^i T_r yes$ in our solution. Let $t_r \in T_r$ s.t. $O_p^i \in t_r$. We know that $T_r \in \mathcal{T}_{current}$ and since $A_j \notin B$, we conclude that $\nexists O_q^j \in t_r$. So condition (2) of the EC constraint is false and thus EC is trivially satisfied.
- *case 3: $A_i \in B, A_j \in B$*
  $A_i = O_p^i T_r yes$ and $A_j = O_q^j T_s yes$ in our solution. Let $t_r \in T_r$ s.t. $O_p^i \in t_r$. T_s and T_r must be strongly conflict free since both are in $\mathcal{T}_{current}$. If $T_s \neq T_r$, then $\nexists$ $O_n^j \in \mathcal{O}$ s.t. $O_n^j \in t_r$. So condition (2) of EC(A_i,A_j) is false and thus EC is trivially satisfied. If $T_s = T_r$, then EC is satisfied since A_j is helping A_i perform T_r.

Next, we show that our assignment satisfies the LC constraints. If $A_i \in B$ then $A_i = O_p^i T_r yes$, and LC1, regardless of the truth value of P, is clearly not violated. Furthermore, it is the case that O_p^i succeeds, since T_r is present. Then the precondition P of LC2 is not satisfied and thus LC2 is not present. If $A_i \notin B$ and $A_i = O_p^i T_r no$, it is the case that O_p^i is executed and, by definition, does not succeed. Then the precondition P of LC1 is not satisfied and thus LC1 is not present. LC2, regardless of the truth value of P, is clearly not violated. Thus, the LC constraints are satisfied by all variables. We can conclude that all constraints are satisfied and our value assignment is a solution to the DyDCSP. □

Theorem II : Given a Class 3 Resource Allocation Problem $\langle \mathcal{A}g, \mathcal{O}, \mathcal{T} \rangle$, $\mathcal{T}_{current} \subseteq \mathcal{T}$ and the corresponding DyDCSP, if an assignment of values to variables in the DyDCSP is a solution, then all tasks in $\mathcal{T}_{current}$ are performed.

proof: Let a solution to the DyDCSP be given. We want to show that all tasks in $\mathcal{T}_{current}$ are performed. We proceed by choosing a task $T_r \in \mathcal{T}_{current}$. Since our choice is arbitrary and tasks are strongly conflict free, if we can show that it is indeed performed, we can conclude that all members of $\mathcal{T}_{current}$ are performed.

Let $T_r \in \mathcal{T}_{current}$. By the **Notification Assumption**, some operation O^i_p, required by T_r will be executed. However, the corresponding agent A_i, will be unsure as to which task it is performing when O^i_p succeeds. This is due to the fact that O^i_p may be required for many different tasks. It may randomly choose a task, $T_s \in T(O^i_p)$, and LC1 requires it to assign the value $O^i_p T_s$yes. The EC constraint will then require that all other agents A_j, whose operations are required for T_s also execute those operations and assign $A_j = O^j_q T_s yes$. We are in solution, so LC2 cannot be present for A_j. Thus, O^j_q succeeds. Since all operations required for T_s succeed, T_s is performed. By definition, $T_s \in \mathcal{T}_{current}$. But since we already know that T_s and T_r have an operation in common, the Strongly Conflict Free condition requires that $T_s = T_r$. Therefore, T_r is indeed performed. □

Given our formalization of dynamic environments, when tasks change, this implies changes only in local constraints of agents, and hence the theorems are able to address dynamic changes in tasks. Class 4 tasks require dynamism in external constraints as well, and this is not handled and an issue for future work.

6 Experiments in a Real-World Domain

We have successfully applied the DyDCSP approach to the distributed sensor network problem, using the mapping introduced in Section 3. Indeed, in recent evaluation trials conducted in government labs in August and September 2000, this DyDCSP implementation was successfully tested on four actual hardware sensor nodes (see Figure 1.a), where agents collaboratively tracked a moving target. This target tracking requires addressing noise, communication failures, and other real-world problems; although this was done outside the DyDCSP framework and hence not reported here.

The unavailability of the hardware in our lab precludes extensive hardware tests; but instead, a detailed simulator that very faithfully mirrors the hardware has been made available to us. We have done extensive tests using this simulator to further validate the DyDCSP formalization: indeed a single implementation runs on both the hardware and the simulator. One key evaluation criteria for this implementation is how accurately it is able to track targets, e.g., if agents do not switch on overlapping sectors at the right time, the target tracking has poor accuracy. Here, the accuracy of a track is measured in terms of the *RMS* (root mean square) error in the distance between the real position of a target and the target's position as estimated by a team of sensor agents. Our results here — assuming a square sensor node configuration of Figure 1.b — are as follows. Domain experts expect an RMS of approx 3 units, and thus they believe these results are satisfactory. Our RMS for Class 1 problems is 0.9 units, and for Class 3 problems, the RMS is 3.5 units.

Table 1 presents further results from the implementation. Experiments were conducted for up to 8 nodes solving Class 3 problems. Each cell in the table presents the number of messages exchanged, the number of sector changes, and in parenthesis, the number of messages exchanged per sector change. For instance, in configuration involving one dynamic target and 6 nodes, agents exchanged 190 messages, for 23 sector changes, i.e., 8.23 message per sector change. More nodes involve more sector changes, since a dynamic target passes through regions covered by different nodes, and the nodes must search for this target to pin down its location.

Table 1. Results from Sensor Network Domain.

Num nodes	4	6	8
one target	57,6 (9.50)	190,23 (8.26)	247,35 (7.05)
two targets	–	91,13 (7.00)	244,29 (8.41)

The key results here are that: (i) The dynamic DCSP algorithm presented in Section 4 is able to function correctly; (ii) increasing number of nodes does not result in increasing numbers of messages per sector change, providing some evidence for scalability of our mapping.

7 Summary and Related Work

In this paper, we proposed a formalization of distributed resource allocation that is expressive enough to represent both dynamic and distributed aspects of the problem. We define four categories of difficulties of the problem and address these formalized problems by defining the notion of Dynamic Distributed Constraint Satisfaction Problem (DyDCSP). The central contribution of the paper is a generalized mapping from distributed resource allocation to DyDCSP. Through both theoretical analysis and experimental verifications, we have shown that this approach to dynamic and distributed resource allocation is powerful and unique, and can be applied to real-problems such as the Distributed Sensor Network Domain.

In terms of related work, there is significant research in the area of distributed resource allocation; for instance, Liu and Sycara's work[5] extends dispatch scheduling to improve resource allocation. Chia et al's work on distributed vehicle monitoring and general scheduling (e.g. airport ground service scheduling) is well known but space limits preclude us from a detailed discussion [1]. However, a formalization of the general problem in distributed settings is yet to be forthcoming. Our work takes a step in this direction and provides a novel and general DyDCSP mapping, with proven guarantees of performance. Some researchers have focused on formalizing resource allocation as a centralized CSP, where the issue of ambiguity does not arise[3]. The fact that resource allocation is distributed and thus ambiguity must be dealt with, is a main component of our work. Furthermore, we provide a mapping of the resource allocation problem to DyDCSP and prove its correctness, an issue not addressed in previous work. Dynamic Constraint Satisfaction Problem has been studied in the centralized case by [6]. However, there is no distribution or ambiguity during the problem solving process. The work presented here differs in that we focus on distributed resource allocation, its formalization and its mapping to DyDCSP. Indeed, in the future, our formalization may enable researchers to understand the difficulty of their resource allocation problem, choose a suitable mapping using DyDCSP, with automatic guarantees for correctness of the solution.

Acknowledgements

This research is sponsored in part by DARPA/ITO under contract number F30602-99-2-0507, and in part by AFOSR under grant number F49620-01-1-0020.

References

1. M. Chia, D. Neiman, and V. Lesser. Poaching and distraction in asynchronous agent activities. In *ICMAS*, 1998.
2. K. Decker and J. Li. Coordinated hospital patient scheduling. In *ICMAS*, 1998.
3. C. Frei and B. Faltings. Resource allocation in networks using abstraction and constraint satisfaction techniques. In *Proc of Constraint Programming*, 1999.
4. Hiroaki Kitano. Robocup rescue: A grand challenge for multi-agent systems. In *ICMAS*, 2000.
5. J. Liu and K. Sycara. Multiagent coordination in tightly coupled task scheduling. In *ICMAS*, 1996.
6. S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *AAAI*, 1990.
7. Sanders. Ecm challenge problem, http://www.sanders.com/ants/ecm.htm. 2001.
8. M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *ICMAS*, July 1998.

Improving Optimality of n Agent Envy-Free Divisions

Stephen W. Nuchia and Sandip Sen

Department of Mathematical & Computer Sciences, The University of Tulsa
600 South College Ave, Tulsa OK 74104, USA
stephen-nuchia@utulsa.edu
sandip@kolkata.mcs.utulsa.edu

Abstract. Division of a resource among multiple agents is a frequent problem in multiagent systems and fair, efficient, and decentralized allocation procedures are highly valued. A division of a good is *envy-free* when every agent believes that its share is not less than anyone else's share by its own estimate. As envy-free procedures are not efficient (in the sense of Pareto optimality) we have previously worked on improving the efficiency of such envy-free division procedures among two agents using models of other agents' utility functions. Though guaranteed envy-free division procedures are available for small number of agents, only approximately envy-free procedures are available for division of a good among an arbitrary number of agents. In this paper, we present an approach by which the outcome of any such approximate algorithms for arbitrary n, or guaranteed algorithms for small number of agents, can be improved in terms of optimality. We propose a two-stage protocol where the first stage identifies possible beneficial exchanges, and the second stage chooses the maximal set of such exchanges that is still envy-free. We map the second stage into a matching problem and present a graph-theoretic algorithm that improves the efficiency of the initial allocations while maintaining the envy-free property.

1 Introduction

A key research issue in agents and multiagent research is to develop negotiation procedures by which agents can efficiently and effectively negotiate solutions to their conflicts [6]. In this paper, we focus on the problem of agents vying for portions of a good. The negotiation process will produce a partition and allocation of the goods among the agents [3,4]. We are interested in both protocols by which agents interact and appropriate decision procedures to adopt given a particular procedure.

We assume that the good being divided is possibly heterogeneous and the preference of an agent for various parts of the good is represented by a utility function. For historical reasons, we will represent the good as a continuously divisible rectangular piece of cake, where we are only interested in the length of the cake.

In an *envy-free* division, each agent believes that it received, by its own estimate, at least as much as the share received by any other agent [1,9]. This also implies that an agent has no incentive to trade its share with anyone else. While such guarantees are indeed extremely useful to bring parties to the discussion table, there is no *a priori* reason why a self-interested agent should be happy with just the share that is most valuable to its estimate in the group when it can possibly get even more. For example, a procedure by which an agent can possibly improve on its share received by an envy-free procedure without losing the guarantee of envy-freeness, would be extremely attractive. Envy-free procedures are also not guaranteed to produce efficient, viz., Pareto-optimal, divisions. This means that it is possible to further re-allocate portions of the cake so that the utility of at least one of the agents is improved without decreasing the utilities of the other agents.

To illustrate this scenario, consider the “divide and choose” procedure, in which one agent cuts the cake into two portions and the other agent gets to choose the portion it wants for itself. The strategy of the cutting agent would be to divide the cake into two portions of equal value by its own estimate. The choosing agent should then choose the portion that is of more value by its own estimate. Thus, both agents would believe they did get the most valuable portion of the cake, and would therefore be envy-free. But there may be portions of the cake in their respective allocations that they can still exchange and further improve their valuations. This problem is only exacerbated with larger number of agents.

In our previous work [7], we presented an approach for two-agent divisions by which agents can use the model of the utility function of the other agent to increase the optimality of the resultant division. The division was guaranteed to be envy-free and was proven to be optimal when such an allocation involved the assignment of a contiguous portion of the cake to one agent. We will refer to this procedure as *2e-opt*.

In the present work, we will focus on the problem of improving the optimality of envy-free divisions among an arbitrary number of n agents. In contrast to our previous work, we will not derive the division from scratch. Instead, we will assume as input an envy-free division of the good among n agents. Such a division with a guaranteed *envy-free* property is know for up to 4 agents; for a higher number of agents, approximate, finite-time procedures have been developed [1]. For our work, we will refer to both the guaranteed and approximate envy-free divisions as envy-free divisions (our discussions will hold for ϵ-envy-free division, where the divisions are envy-free within a bound).

We propose a two-stage reallocation process to improve the optimality of division. In the first stage, every pair of agents divide up their allocated portion of the cake using *2e-opt* and identify beneficial exchanges. In the second stage, a set of such exchanges is chosen that maximizes the total utility value without evoking the envy of any one agent. The procedure presented has the property of maximizing the optimality of the solution starting from the given envy-free division.

To set the context for this work, we will summarize the requisites of fair division processes, the framework for negotiation, and the algorithm *2e-opt* from our previous work. We will then present the two stages of the protocol mentioned above and discuss planned improvements.

2 Division of a Good

We concern ourselves with the problem of dividing up a good between multiple individuals. We will assume that the solution procedure is of a decentralized nature. This means that the agents will be required only to abide by a protocol by which the division is to be made. They can freely choose any strategy to use to determine their actions within the accepted protocol.

For example, a protocol might be that every agent submits a sealed bid for the good. After every bid is collected, the good is divided up among the bidders in proportion to their bids. Our assumption is that once the agents agree to such a protocol, they are free to choose their bids following any strategies they adopt. We require, however, that agents will agree to the division of the good as specified once they have placed their bids.

From a designer's point of view the choice of a protocol provides a platform for agents to negotiate an agreeable division. The choice of a strategy will be dictated by concerns for arriving at a preferred share of the good being divided. The protocol designer should then provide protocols which can be used by agents to successfully negotiate agreeable divisions with reasonable computational costs. We will now look at properties of divisions that can make them agreeable to self-interested agents.

2.1 Desired Characteristics of Divisions

We assume that a single divisible good is to be divided among n agents. The following criteria have been espoused as desirable characteristics of decision procedures or outcomes from such procedures [1]:

Proportional: Each agent believes that it received, by its own estimate, at least $\frac{1}{n}$ of the goods being allocated.

Envy-free: Each agent believes that it received, by its own estimate, at least as valuable a share as that received by any other agent. This also implies that an agent has no incentive to trade its share with anyone else.

Equitable: A solution, i.e., a partition of the good among the n agents, is equitable, when the share received by each agent is identical in terms of their individual utility functions.

Efficient: A solution is said to be Pareto optimal or efficient if there is no other partition which will improve the perceived share of at least one agent without decreasing the perceived share of any other agent.

2.2 Envy-Free Divisions with Improved Efficiency

We assume that the continuously divisible good is possibly heterogeneous and the preference of an agent for various parts of the good is represented by a utility function. Though traditionally represented as a rectangular piece of cake, only the length dimension plays a role in the algorithm. Though we present the *2e-opt* procedure from our previous work [7] here for completeness, at first glance the reader can simply use the results from the following theorems to facilitate understanding of the discussion about the n-person protocol in the later sections of this paper.

The utility to the ith agent of a piece of the cake cut between points a and b (where $a < b$) is given by $\int_a^b U_i(x)dx$, where $U_i(x)$ is the utility function of the ith agent. Without loss of generality, we will assume that the first agent is having a model, $\overline{U_2}$, of the utility function of the second agent. This model need not be accurate, but the better the model, the more benefit the modeling agent stands to gain by using it.

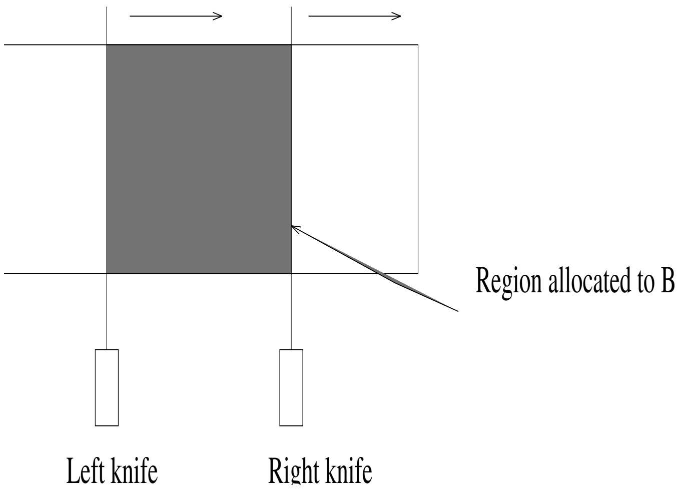


Fig. 1. Our augmentation of Austin's procedure with the modeling agent moving two knives.

The *2e-opt* procedure we now describe is derived from Austin's moving knife procedure [1]. In the *2e-opt* procedure [7], the modeling agent A holds two knives parallel to the side edges of the cake and then move them to the right allowing for wrap around. Let the positions of the left and right knives at time t be l_t and r_t respectively. l_0 and r_0 determine the initial region offered to B. The knives stop at time $t = T$ when the right knife reaches the original position of the left knife, and the left knife reaches the original position of the right knife. The agent B then chooses a time $\tau \leq T$, and the portion of the cake in between l_τ and r_τ (with wrap-around if needed) is given to B, with the rest of the cake going to agent A (see Figure 1).

It can be shown that B can always produce an envy-free division for itself if it calls "cut" at an appropriate time irrespective of how A moves the knife. B, however, can be resentful as it can presume that the advantage of moving the knife allows A to obtain a super-equitable share for itself, which guarantees a sub-equitable share for B.

We now specify the choice of the initial region, the choice of the target region that the modeler wants the other to choose, and a way of moving the knife such that this target region is most likely to be chosen by the other agent.

Initial region selection: The initial knife placements should satisfy $\int_n^m U_A(x)dx = \frac{1}{2}\int_0^L U_A(x)dx$, where L is the length of the cake. It is also desirable that the region in between does divide the cake in half for agent B. Otherwise B may find it beneficial to choose $t = 0$ (if, according to B's estimate, the region (n, m) is the most valuable) or $t = T$ (if, according to B's estimate, the region (m, n) is the most valuable). But a contiguous region that satisfy both these conditions may not be found in general. So, the initial placement (m, n) should also satisfy the following condition:

$$(m, n) = \arg\min_{y,x} \left| \int_x^y \overline{U_B}(z)dz - \frac{1}{2}\int_0^L \overline{U_B}(z)dz \right| .$$

For ease of exposition, if $y < x$, we use $\int_x^y f(z)dz$ to represent $\int_x^L f(x)dx + \int_0^y f(z)dz$.

Target region selection: The target region must result in an envy-free division. Eliminating beginning and end points, the target region should be selected from the following set:

$$Z = \{(x,y) :(\int_x^y \overline{U_B}(z)dz > max(\int_{l_0}^{r_0} \overline{U_B}(x)dx, \int_{l_T}^{r_T} \overline{U_B}(x)dx)) \wedge(\int_x^y U_A(z)dz < \frac{1}{2}\int_0^L U_A(z)dz)\}. \tag{1}$$

From the regions in Z, we filter out those regions that are of least value to A: $Y = \arg\min_{(x,y)\in Z} \int_x^y U_A(z)dz$. From the regions in Y, A can select those regions which are estimated to be of most value to B: $X = \arg\max_{(x,y)\in Y} \int_x^y \overline{U_B}(z)dz$. If X is non-empty, then A's goal is to select one of its elements and then move the knives such that the corresponding region is the most attractive offer received by B. If X is empty, however, A will have to be satisfied with half of the cake.

Moving the knife: While moving between point pairs identified in the previous paragraph, the knife locations (u, v) should satisfy the following inequality: $\int_u^v \overline{U_B}(x)dx < \frac{1}{2}\int_0^L \overline{U_B}(x)dx$. For regions where A's utility is high, the spacing between the knives will be reduced to make that region non-envy-free for B.

An instance of the above procedure is shown in Figure 2. We present some properties of this decision procedure (for proof see [7]):

Theorem 1 *Our scheme dominates Austin's procedure with respect to efficiency of allocations.*

Theorem 2 *If a Pareto-optimal allocation for a problem involves a contiguous region, our proposed scheme will select it when given an accurate agent model.*

3 Optimality of n-Agent Envy-Free Divisions

Now, we present our two stage protocol for improving the optimality of a given n-person envy-free division of a continuously divisible good. In the first stage of this procedure, each pair of agents identify possible exchanges by dividing each of their shares using *2e-opt*. We define a graph mapping $G = (V, E)$ of this scenario where each vertex $v \in V$ of the graph corresponds to an agent, and the weight w_{ij} on the undirected edge e_{ij} between vertices i and j correspond to the net gain in utilities for a plausible envy-free exchange between agents i and j. An exchange is plausible if both agents gain from it. If there is no plausible exchange between two agents, there is no edge between the two corresponding vertices in G (or the corresponding edge weight is zero).

The second stage of the exchange protocol then involves finding the set of such edges with maximum weight such that no agent is envious of the allocations received by other agents after the exchanges, corresponding to the chosen set of edges, are carried out. Any and all agents who are envious of one or more such exchanges can veto the corresponding

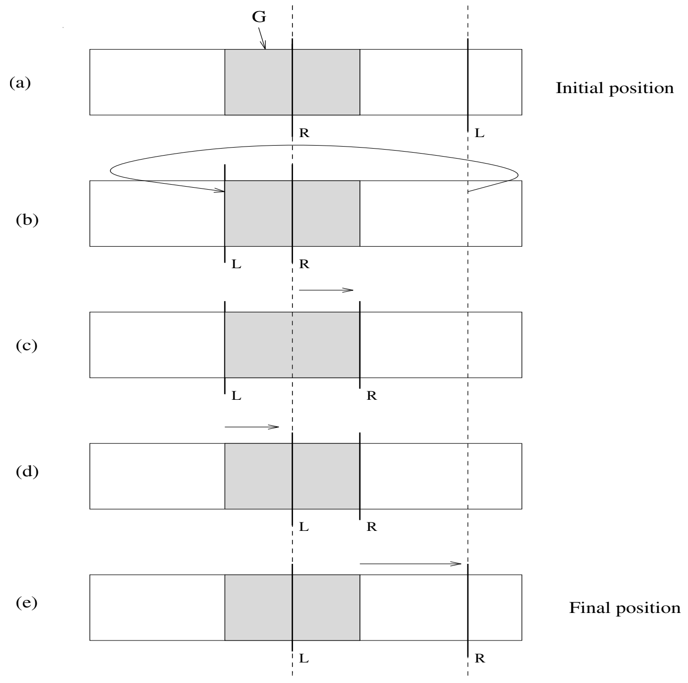


Fig. 2. Knife-moving procedure snapshots (G: target region; L,R: left, right knife).

edges from the chosen set. The protocol consists of first finding a maximal weight edge set such that no more than one edge in the set is incident on any vertex of the graph. This means that an agent will participate in at most one exchange. Next, agents are asked to evaluate the resultant allocations for envy. If the resultant allocation produces envy, the algorithm chooses the next best set of edges. Each iteration of the protocol produces the maximal possible improvement in optimality under the restriction that a single round of pairwise exchanges (identified by *2e-opt*) is executed. The protocol should be iterated until no further improvement is feasible or the available time is exhausted.

In the following we describe these two stages of the protocol in more detail.

3.1 Finding Pair-Wise Envy-Free Exchanges

In this stage, each pair of agents determine if there is a plausible envy-free exchange between them. We illustrate this process by considering two agents A and B. Let us assume that an envy-free division has produced an allocation of portion X to agent A and portion Y to agent B. Now, *2e-opt* is applied on both X and Y dividing it into portions (X_A, Y_B) and (Y_A, Y_B) respectively. This means that if X were to be the only portion to be divided up between A and B, then (X_A, X_B) would be an envy-free division with A receiving X_A and B receiving X_B, i.e., X_A (X_B) is the more valuable portion of X in A's (B's) estimate. Note that (X_A, X_B) is not the optimal envy-free division possible, but the one that is computed by *2e-opt* (our prior work [7] shows that there is no finite,

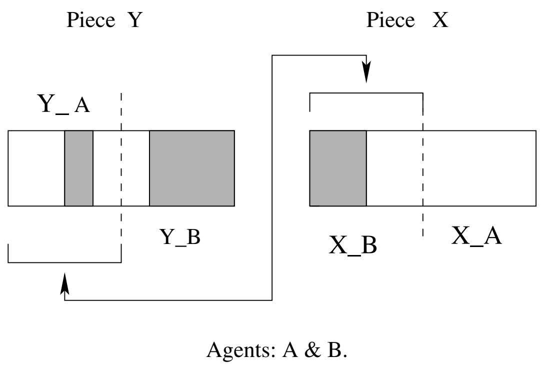


Fig. 3. A plausible envy-free exchange between agents A and B.

optimal, envy-free division possible in general). Similar reasoning holds for the portion Y. There exists a plausible exchange between A and B if A believes Y_A is more valuable than X_B, and B believes that X_B is more valuable than Y_A. This process is illustrated in Figure 3. Assuming this exchange is envy-free, the weight in the graph on the edge between A and B will be (A's valuation of Y_A - A's valuation of X_B) + (B's valuation of X_B - B's valuation of Y_A).

This process can be repeated for each pair of agents. Hence in $O(n^2)$ time, where n is the number of agents, we can compute the graph structure mentioned before.

3.2 Choosing Optimal Envy-Free Exchanges

In the second stage, the set of edges with maximal weight and satisfying the constraint that no agent participates in more than one exchange and that the corresponding exchanges do not result in envy, has to be chosen. In the following we present a graph-theoretic algorithm for selecting the set of edges of maximal weight such that no agent partakes in multiple exchanges. We also present the procedure for selection of all such edge sets in the decreasing order of their total weight. We traverse down this order as long as the previous selections result in envy. Once a selection is found that is envy-free, we have the best optimality improvement (subject to the use of *2e-opt* in determining plausible exchanges) that is also envy-free given the starting envy-free partition.

The problem of finding improved envy free allocations is related to the maximum weight (vertex) matching problem in the following way. A matching in a simple general graph is a subset of the edges in which no two edges share a vertex. A matching can be viewed as partitioning the vertices into a set of disjoint pairs and a set of non-paired vertices. The edges in a matching and the vertices adjacent to those edges are said to be *matched*, edges and (especially) vertices that are not matched are said to be *free*. Formally, a matching in a graph $G = (V, E)$ is a subset $M \subseteq E$ such that the maximum degree in the graph (V, M) is one.

The *weight*, $w(M)$, of a matching M is the sum of the weights of the matched edges. A maximum weight matching in G is a matching having the largest weight among all possible matchings in G. A well-known algorithm exists with time complexity of $O(n^4)$

for finding a maximum weight matching in an arbitrary graph [2]. Other more recent algorithms have time complexities of $O(n^3)$ and $O(nm \log n)$, where m is the number of edges and n is the number of vertices in the graph [5]. These algorithms “grow” a matching by identifying what are called *augmenting paths*; the best such path with respect to a matching merged with that matching gives a new, better matching. (The new matching is the *symmetric difference* of the path and the old matching.) The algorithms differ in the details of how a best augmenting path is found.

We have an incremental version of this algorithm that will find a new max weight matching after deleting from the graph an edge found in some max weight matching. In the incremental algorithm the definition of an augmenting path must be relaxed to allow even-length alternating paths and to treat specially paths joining the endpoints of the deleted edge. Using the incremental algorithm is approximately a factor of $\frac{n}{6}$ faster than recomputing the max-weight matching from scratch, though our search works correctly either way. An abbreviated proof of the incremental algorithm given in section A. Analysis of the data structure options for the algorithm is the subject of a future work.

A matching corresponds to a reallocation of the goods and this new allocation may not be envy free. A reallocation is envy free if no participant envies any other participant assuming that the exchanges have taken place. This hypothetical envy is an arbitrary (anti-reflexive) relation on the vertices but, when considering the envy relations defined for two different allocations, the truth value of “u envies v” will not change if the portions allotted to u and v are unaltered.

We search for the independent set of exchanges maximizing total utility without envy using a generate-and-test strategy. Candidate matchings are generated in approximately best-to-worst order and tested in monotonically non-decreasing order. Thus the first matching found to be envy free is a best envy-free matching. The generator is exhaustive but not efficient: it is guaranteed to find all feasible matchings but may visit some more than once.

One further structural property of the envy relation will be useful. The maximum incremental satisfaction available to any participant is a known quantity. Suppose in an instance of the problem some participant k has a potential exchange (an edge) that is not part of the matching being evaluated. It is possible that k would envy (veto) some exchange (edge) based on his current satisfaction but would approve of that exchange if he enjoyed his maximum satisfaction. On the other hand, if some edge would be vetoed by k even when k's best edge was included in the matching then the vetoed edge can not be in any feasible matching for that graph. Incorporating this observation, we now outline the search algorithm.

Begin by finding a best matching M_0 in $G = (V, E)$. Initialize a work queue with the tuple $(\mathcal{P}(E), M_0, w(M_0))$ representing the virtual list of subproblems generated from G — that is, the set of all potentially feasible matchings in G. With the work queue prioritized by the weight of the matchings, largest first, the leading weight is in fact a tight upper bound on the weight of all not-yet-tested matchings.

We now loop until the work queue is exhausted or a feasible matching is discovered. For each tuple $(\mathcal{P}(E_i), M_i, w(M_i))$ extracted we begin by evaluating the envy-free predicate. If the matching is envy free we have found a best envy-free matching and may stop.

If some of the edges in M_i are vetoed we go on to check whether any of those vetoes are permanent in the sense described above. If so we reduce the problem to $(\mathcal{P}(E_i'), M_i', w(M_i'))$ by deleting the permanently vetoed edges from the set representations of E_i and M_i, computing a new max weight matching M_i' in the new subgraph (V, E_i'). Depending on the number of edges deleted, M_i' may be computed incrementally or recomputed from scratch. In any case, the reduced graph contains (or represents) all possible envy-free matchings contained in the original graph. Since in general $w(M_i') \leq w(M_i)$ we insert the reduced tuple into the work queue and loop.

At some point we withdraw an entry from the queue with no permanently infeasible edges. If that matching is nevertheless infeasible (or if we wish to enumerate all feasible matchings) we need to put back into the work queue all the not-yet-tested members of the set of potential matchings represented by that tuple. With M_i a maximum weight matching in the positively weighted edge set E_i, the set

$$\cup_{e \in M_i} \mathcal{P}(E_i - \{e\}) \subseteq \mathcal{P}(E_i) - M_i$$

contains all matchings in E_i except M_i. For every edge e_j in M_i we generate a new work queue entry $(\mathcal{P}(E_i - \{e_j\}), M_{i,j}, w(M_{i,j})$ where $M_{i,j}$ is the maximum weight matching in $(V, E_i - \{e_j\})$ computed incrementally from M_i.

When inserting new entries into the work queue it may be worthwhile to eliminate redundancies in some way. We plan to examine the data structures and time/space tradeoffs associated with various redundancy-reduction strategies.

4 Discussion

In our prior work we presented an algorithm that produced optimal envy-free divisions of continuously divisible goods between two agents under certain assumptions of utility functions. In this paper, we have presented an interesting application of this procedure to improving the optimality of any envy-free division between an arbitrary number of agents. The two-stage protocol we have proposed first identifies all pair-wise envy-free exchanges and then utilizes a graph-based generate-and-test matching algorithm that selects the optimal set of such exchanges that will still produce envy-free allocations.

Our proposed protocol in this paper uses the algorithm presented in our previous work [7] for generating the envy-free exchanges between pairs of agents. The current protocol, however, will work with any algorithm that generates envy-free exchanges between two agents. We have recently worked on a recursive version of our previous algorithm that will be an anytime algorithm to generate the best possible exchange between two agents given a limit on the number of recursive invocations [8] (the limit is actually the number of cuts allowed on the cake; in the worst case, we will need an infinite number of cuts to generate an optimal exchange, which is not a feasible solution in practice). We can incorporate such a recursive version or any other pairwise exchange mechanism that improves optimality as the first stage of the protocol presented here.

References

1. Steven J. Brams and Alan D. Taylor. *Fair Division: From cake-cutting to dispute resolution.* Cambridge University Press, Cambridge: UK, 1996.
2. J. Edmonds. Maximal matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, B-69:125–130, 1965.
3. Michael N. Huhns and Anuj K. Malhotra. Negotiating for goods and services. *IEEE Internet Computing*, 3(4), 1999.
4. J. Robertson and W. Webb. *Cake Cutting Algorithms.* A. K. Peters, Natick, MA, 1998.
5. Kenneth H. Rosen, editor. *Handbook of Discrete and Combinatorial Mathematics.* CRC Press, Boca Raton, LA, 2000.
6. J. S. Rosenschein and G. Zlotkin. Designing conventions for automated negotiation. *AI Magazine*, pages 29–46, Fall 1994.
7. Sandip Sen and Anish Biswas. More than envy-free. In *Working Notes of the AAAI-99 Workshop on "Negotiation: Settling Conflicts and Identifying Opportunities"*, pages 44–49, 1999. AAAI Technical Report WS-99-12.
8. Sandip Sen and Rajatish Mukherjee. Negotiating efficient envy-free divisions. In *2001 AAAI Fall Symposium on Negotiation Methods for Autonomous COoperative Systems*, pages 149–154, Menlo Park, CA, 2001. AAAI Press. AAAI Tech Report FS-01-03.
9. I. Stewart. Mathematical recreations: Division without envy. *Scientific American*, January 1999.

A Proof of Incremental Matching Algorithm

A condensed proof of our algorithm for finding maximum weight matchings (MWM) after deleting an edge is presented here. An expanded version of this proof is available from the authors.

Let $A \oplus B$ denote the *symmetric difference* $(A \setminus B) \cup (B \setminus A)$ between two sets. We have chosen to use the symbol $\oplus$ to emphasize the isomorphism between set symmetric difference and binary exclusive OR.

A path P in G is *alternating* with respect to a matching M (also in G) if each vertex of degree two in P is adjacent to exactly one edge in M. The *weight* of an alternating path P with respect to a matching M is the sum of the weights of the edges in P when those in M are counted negatively. For an edge set (or graph) E let $w(E)$ be the sum of the edge weights in E. Then the weight of P with respect to M is $w_M(P) = w(P \setminus M) - w(P \cap M)$. Alternating cycles are defined similarly.

A.1 The Difference Component Representation

Let A and B be matchings in a graph G and consider the subgraph induced in G by the symmetric difference $A \oplus B$. We refer to the collection of components of that subgraph[1] as the *component representation* of the difference or DCR and denote it by DCR(A, B).

Theorem 3 DCR(A, B) *is a set of paths and cycles in G and is empty only if* $A = B$.

[1] Strictly speaking, the collection of edge sets of the components.

Proof. Any subset of a matching is a matching and $A \oplus B$ is the union of two disjoint subsets of matchings. Each subset can contribute no more than one edge adjacent to any vertex and therefore no vertex in $A \oplus B$ has degree greater than two. □

Let $A \neq B$ and let C be a component of their DCR . Furthermore, let

$$S = \bigcup_{X \in Y} X \quad \text{for some} \quad Y \subseteq \text{DCR}(A, B).$$

Then:

Property 1 *C is alternating with respect to both A and B.*

Property 2 *$A \oplus S$ and $B \oplus S$ are matchings in G.*

Proof. Property 1 is proved using the same reasoning as theorem 3. Proof of property 2 involves a case analysis over the possible degrees of vertices in A, B, and S, all but one case being trivial.

Assume the edge $\{u, v\}$ is in A and the edge $\{v, w\}$, with $u \neq w$, is the only edge in S incident on v. This second edge must be in B since $S \subset A \cup B$ and it cannot be in matching A.

Since the second edge is in B the first edge cannot be and thus both edges are in $A \oplus B$. But then u and w are connected in the subgraph induced by $A \oplus B$ and so the edges connecting them must be in the same component of the DCR . Therefore it is not possible that one is in S and the other is not, contradicting the assumption. □

Note that A and $\text{DCR}(A, B)$ determine a family of matchings, and that family includes B. Let M, M', etc. denote arbitrary matchings in the family determined by A and $\text{DCR}(A, B)$. In particular, let $M' = M \oplus S$.

If the graph is edge weighted then:

Property 3 $w(A \oplus S) = w(A) + w_A(S)$.

Property 4 $w_A(S) = -w_B(S)$.

Theorem 4 *If A and B are maximum weight matchings in G then all components of $\text{DCR}(A, B)$ have zero relative weight:*

$$\forall C \in \text{DCR}(A, B) \quad w_A(C) = w_B(C) = 0.$$

Proof. Assume some component C has non-zero weight. Then by property 4 either $w_A(C) > 0$ or $w_B(C) > 0$ and so

$$w(A \oplus C) > w(A) \quad \text{or} \quad w(B \oplus C) > w(B),$$

contradicting the assumption. □

A.2 Graphs Differing by One Edge

Consider a graph $G = (V, E, w)$ and $e \in E$. Let subgraph $G' = (V, E', w)$ where $E' = E - e$. Assume $w(e)$ is sufficiently large that e is a member of some MWM M in G. Furthermore let M' be an arbitrary MWM in G': necessarily $e \notin M'$.

Theorem 5 DCR(M, M') *consists of one component C_e containing e and having non-positive weight relative to M. All other components have zero relative weight.*

Proof. We begin by observing that $e \in M \oplus M'$ so C_e must exist. If $w_M(C_e)$ were positive we would have $w(M \oplus C_e) > w(M)$ but the weight of M is assumed to be maximal.

If for some $C \neq C_e$ we have $w_M(C) > 0$ then $w(M \oplus C) > w(M)$, a contradiction.

Because the components of DCR(M, M') are disjoint $M' \oplus C$ does not contain e; it is a matching in G'. If $w_M(C) < 0$ then by property 4 we have $w_{M'}(C) > 0$ and so $w(M' \oplus C) > w(M')$, another contradiction. □

A.3 The Algorithm

We have reduced the problem of finding an MWM in a graph with a known MWM after removing one edge found in that MWM to the problem of finding an "appropriate" alternating path or cycle containing the edge to be deleted. Rather than directly characterize of the path sought we will describe the search and prove that the result gives us a maximum weight matching.

The singleton path $\{e\}$ is alternating with respect to M so its weight, $w_M(\{e\}) = -w(e)$, is a lower bound on the weight of the path or cycle we seek. Suppose some alternating path or cycle C containing e exists with a larger relative weight: then $C - e$ consists of at most two paths, alternating and having non-negative weight with respect to M.

Search, for example[2] by depth-first exploration, beginning at one endpoint of e and avoiding repeat visits to vertices while alternating edges with respect to M. At each vertex encountered that is not matched in $M - e$ and each time the path length (search tree depth) is even, note the relative weight of the path so far. Prune the tree and separately note the relative weight when the second endpoint of the deleted edge is encountered. If at any time a path weight equal to $w(e)$ is noted the search may be aborted: since $w_M(C_e) \leq 0$ this path weight is maximal.

The result of the first search is a best path P_1 from the first endpoint of e and a best path P_{12} joining[3] the endpoints of e. Repeat the search beginning at the second endpoint of e and avoiding vertices appearing in P_1; let P_2 denote a best path found in the second search. If a path with relative weight equal to $w(e) - w_M(P_1)$ is noted the search may again be stopped early.

Theorem 6 *Let $\widehat{C} = P_{12} \cup \{e\}$ if $w_M(P_{12}) > w_M(P_1 \cup P_2)$ and $\widehat{C} = P_1 \cup P_2 \cup \{e\}$ otherwise. Then $\widehat{M} = M \oplus \widehat{C}$ is an* MWM *in G'.*

[2] Any suitable optimization from the fast MWM algorithms may be used.

[3] P_{12} will be empty if no alternating cycle in G with respect to M contains e.

Proof. If C_e is a path then $C_e - e$ is a pair of paths (one or both may be empty), one anchored at each endpoint of e. In that case the paths may be of odd or of even length but each will start with an edge not in M. If on the other hand C_e is a cycle then $C_e - e$ is an odd-length path beginning and ending at the vertices of e.

Let $\widehat{M} = M \oplus \widehat{C}$. Since the characteristics of C_e enumerated above are consistent with those of the paths considered by the search algorithm we know $w_M(\widehat{C}) \geq w_M(C_e)$ and therefore $w(\widehat{M}) \geq w(M')$. It remains to show that $\widehat{M}$ is a matching in G'.

First we observe $e \notin \widehat{M} \subseteq E$ so $\widehat{M} \subset E'$.

Because we prune the search if the second endpoint is encountered, we know that the degree of the endpoints of e in $\widehat{C}$ is no more than two. Since edges incident on vertices already visited are not added to the path, no other vertex can have degree more than two. By construction then, $\widehat{C}$ is a path or cycle alternating with respect to M.

The degree of each interior (degree two) vertex in $\widehat{C}$ is unchanged in computing $\widehat{M} = M \oplus \widehat{C}$. Assume $\widehat{C}$ is not a cycle; then two of its vertices have degree one. These vertices correspond to the accepting states of the search algorithm and thus either either have degree zero in M (the odd length case) or are adjacent to an edge that is in both M and $\widehat{C}$ (the even case). In the first case the vertex has degree one in $\widehat{M}$ and in the second its degree there is zero.

All vertices not appearing in $\widehat{C}$ will retain their degrees from M in $\widehat{M}$. Thus $\widehat{M}$ is a matching in G' and since $w(\widehat{M}) \geq w(M')$ it is a maximum weight matching in G'. □

Trustworthiness of Information Sources and Information Pedigrees

Wei Liu[1,2] and Mary-Anne Williams[1]

[1] Business and Technology Research Laboratory
The University of Newcastle, Newcastle, NSW 2308, Australia
eFax & eVoiceMail: +1-650-745-3301
{wei,maryanne}@ebusiness.newcastle.edu.au
[2] also at School of Computer Science and Information Technology
RMIT University, Melbourne, VIC 3000, Australia
Tel: +61-3-99259500 Fax: +61-3-99621617
wei@cs.rmit.edu.au

Abstract. To survive, and indeed thrive, in an open heterogenous information sharing environment, an agent's ability to evaluate the trustworthiness of other agents becomes crucial. In this paper, we investigate a procedure for evaluating an agent's trustworthiness as an information source. By separating the procedure into *competency analysis* and *sincerity analysis*, we are able to deal with complicated cases involving the passing-on of information, where the same information may reach a receiver agent via different routes. In order to keep information about the source agent we use an *information pedigree* as a means to maintain the history of communicated information. Our evaluation of trustworthiness can be employed to drive data fusion, weighted knowledge base merging, and multiple source conflict resolution.

1 Introduction

Agent-oriented software engineering has gained tremendous attention in recent times[10]. Its practical and theoretical elements are proving to be helpful in understanding and modeling complex systems, and as a result have lead to new and innovative designs.

A dominant approach in the agent literature stems from viewing agents as software modeled on human qualities. In this approach agents are ascribed mental states (beliefs, desires etc), have personalities (benevolent, malicious etc) and are able to make decisions and act autonomously (i.e. without human intervention). The most promising attribute of agents is that, in addition to the ability to stand alone, make decisions and perform on their own as individual entities, they can interact with each other, establish (cooperative or competitive) relationships, thus forming so-called agent societies. Some researchers go even further by exploring the idea of a digital city[1], which hosts citizen agents under

[1] Digital City: http://www.digitalcity.gr.jp/virtualcommunity/tourguide.html

the governance of police agents and military agents, and so forth. In such virtual agent societies, analogous to physical human societies, it is necessary to consider the possibilities of fraud, being cheated, and being deceived during information exchanges. This raises the issue and fuels the growing interest in *trust*.

The interest in a formal theory of trust for the purpose of agent research can be traced back to the early 90's[9]. Furthermore, there has been a resurgence in recent years inspired by *The First International Workshop on Deception, Fraud and Trust in Agent Societies* in 1998.

Despite the broad use of the term "trust" in network security, cryptography and e-commerce, it is rarely generally defined in the agent literature. One reason for this is that the notion itself is highly context-sensitive and can be given a variety of readings; one can define it for specific applications but not easily for the general case. Elofson[5] attempted to give a composite definition of trust after studying several perspectives on the meanings of trust:

> Trust is the outcome of observations leading to the belief that the actions of another may be relied upon, without explicit guarantee, to achieve a goal in a risky situation.

From our vantage point, this definition is a good definition because it emphasizes the belief aspects of trust, the risk involved in trust and the close relationship between trust and delegation. However, there is an important element missing, one that we are particularly interested in, that is the "reliance" on the information an agent conveyed[2].

It is important to clarify the phrase *trust of an information source*. This is different from trust of another agent's capability and responsibility of carrying out certain delegated tasks. It is about the credibility of the information delivered by the source agent. For example, consider the following questions:

1. Shall I trust the informant, is it possible that he is lying to me?
2. Is he competent in the subject matter, how can I know that he is not telling me something stupid?

Thus, an agent's trustworthiness as an information source is evaluated using the credibility of the information conveyed.

In this paper, we focus on how an agent can obtain and evaluate the trustworthiness of an information source. Two essential constituents belief are identified in section 2, namely, *competency* and *sincerity*. In section 3, we propose the corresponding evaluation procedures. By evaluating sincerity and competency separately, and in addition to the *information's pedigree*, we are able to analyze and model the process of passing-on information in section 4. Using an example, section 5 demonstrates an application of evaluating trustworthiness during the process of resolving conflicting information. The paper concludes with section 6.

[2] But on the other hand, communication can be thought of as a special form of action performed by the trustee agent, which is perceived by the trustor agent. One could argue Elofson definition covers the special case of defining trust on information sources.

The work presented in this paper is currently being incorporated into a social belief revision agent who is capable of revising both its social knowledge base and its domain knowledge base (i.e. revising information from multiple sources). In addition the belief revision agent is capable of interacting with other agents to carry out multi-agent belief revision[12][13][14].

2 Essential Constituent Beliefs for Trusting Information Sources

Castelfranchi and Falcon[2] pointed out that the degree of trust is a function of the subjective certainty of the *pertinent beliefs*, namely, the cognitive constituents that are essential to establish trust with another agent. These include competence belief, disposition belief, dependence belief, fulfillment belief, willingness belief, persistence belief and self-confidence belief. The component beliefs are identified and evaluated to help the decision making in delegating certain tasks to a trustee, not for the purpose of determining the reliability of a trustee as an information source. However, it does shed light on how to decompose the problem of evaluating trustworthiness into tractable components. Taking a similar approach, in this section, we identify the essential constituent beliefs for trusting an information source.

The communicative act considered here for exchanging information is most appropriately described as an *inform* act, using performatives defined in FIPA ACL[3]. Provided that the communcation is successful, the result of the inform act is that the receiver successfully recieved the information the sender intended to send. In other words, at a stage where an agent received some information from another agent, the action of infoming has already been performed by the trustee (i.e. the sender). Therefore, the evaluation on disposition, willingness, fullfillment and persistence belief is no longer necessary, as these are beliefs regarding the extent to which the delegated action (i.e. informing in this case) is to be carried out. This is the major difference between trust on delegation and trust of information sources, when trust of information sources is considered as a special case of trust on delegation. What is left are the competence belief and the dependence belief, where dependence belief is closely related to sincerity belief defined in the list below.

According to Demolombe[3], a sender agent's trustworthiness on delivering correct information is discussed on the basis of four elementary properties:

1. Sincerity: agent A is sincere with respect to agent B when he believes what he says to B.
2. Credibility: agent A is credible when what he believes is true in the world.
3. Co-operativity: agent A is co-operative with respect to agent B if he tells B what he believes.

[3] FIPA: Foundation of Intelligent Physical Agent: http://www.fipa.org , ACL: Agent Communcation Language.

4. Vigilance: agent A is vigilant with respect to p if, whenever p is true in the world, then A believes p.

Not surprisingly, philosopher John Hardwig[8] pointed out that, the reliability of a receiver's belief depends on the reliability of the sender's character, both his moral character and his epistemic character. The moral character is mainly the agent's truthfulness, while the epistemic character involves his level of competence, conscientiousness and epistemic self-assessment.

It is not hard to see the two essential pertinent beliefs that both Demolombe and Hardwig agree on here are, the sincerity and the competency belief. Furthermore, the discussion earlier in this section using Castelfranchi and Falcon's approach confirms the same decomposition.

Observation The sincerity and the competency belief are the necessary constituents in the evaluation of a social cognitive agent's trustworthiness as an information source. Although these two are not the only ones required, they are sufficient in the sense that most other possible constituents fall into these two categories.

What about the *cooperativity* and *vigilance* proposed by Demolombe[3]? Should they be in the set of the component beliefs for trusting an information source?

As a necessary condition for an agent to be credible, *vigilance* falls into the broader category of competency.

Cooperation and trust have a strong reciprocal relationship with each other. That is, a trusting climate encourages cooperation, and a positive cooperative outcome in turn enhances the level of trust for all parties involved. Marsh[16] claims that cooperation can be classified according to the incentive as cooperation towards mutual rewards, cooperation based on communal relationships and cooperation that emerges from coordination. No matter what incentive makes an agent cooperative or not, the result of cooperativity in information exchange is a matter of whether the receiver (trustor) obtained some information from the sender (trustee) or not. Generally, agents who are not cooperative will either provide no information or wrong/misleading information. Whereas, agents who are cooperative will provide information truthfully. As a consequence cooperativity analysis for information exchange can be considered as an important factor that may affect the degree of sincerity. It then becomes part of the sincerity analysis but not necessarily an independent component belief in its own right.

Assumption By accepting the separation of trustworthiness into competence and truthfulness beliefs, from now on, we can assume the trustworthiness of information sources are scalable and comparable.

Therefore, by interpreting trustworthiness in terms of agents' competency and sincerity, we are able to model the following types of agents:

– Q_1: Trustworthy Agents (agents who are both truthful and credible)

- Q_2: Honest Ignoramus (agents who are honest but not credible)
- Q_3: Dishonest Expert (agents who are credible and not honest)
- Q_4: Nonsense Agents (agents who are neither truthful nor credible)

3 Trustworthiness and Degree of Beliefs

In the fields of data fusion, weighted knowledge base merging, mulitple source conflict resolution and multi-agent belief revision, the value of trustworthiness is always assumed to be given. In this section, based on the observation in section 2 above, we present two ways of using trustworthiness, one is that the degree of trustworthiness t is used as a single value, the other is that the degree of sincerity (s) and the degree of competency (c) are used separately with the degree of beliefs δ.

3.1 Degree of Trustworthiness

Given the value of an information source's competency and sincerity, the total trust on an information source T can be computed using an evaluation function $\tau(s,c) : S \oplus C \rightarrow T$, which maps a degree of sincerity $s \in S$ and a degree of competency $c \in C$ to the trustworthiness of an information source $t \in T$. $s, c, t \in [0,1]$, with 1 the most truthful, credible and trustworthy respectively, 0 the least. The greater the value of t, the more trust an agent has in an information source.

$$\tau(s,c) = \begin{cases} 1, & \text{if } s = c = 1 \\ 0, & \text{if } s = c = 0 \\ c, & \text{if } s = 1 \\ s, & \text{if } c = 1 \\ s \oplus c, & \text{otherwise.} \end{cases}$$

and $\forall y \exists z, t_x(y) \leq t_x(z)$ or $t_x(z) \geq t_x(y)$

Subscripts and variables (e.g. x, y, z...)are used to indicate trustee and trustor. For example, $t_x(y)$ is agent x's faith in agent y as an information source. Similarly, we can have $s_x(y)$ and $c_x(y)$ for agent x's evaluation of agent y's sincerity and competency respectively.

Three simple operators that can be used for $\oplus$ are:

- $\tau_{mulitply} = s \times c$
- $\tau_{min} = min(s,c)$
- $\tau_{ssf} = 1 - (1-s)(1-c)$
  $= s + c(1-s)$
  $= c + s(1-c)$

The multiplication $\tau_{mulitply}$ and the minimum operator τ_{min} are straightforward, and need little explanation. The third function τ_{ssf} is the so-called *Bernoulli's* *rule of combination*, which is a special case of Dempster-shafer's rule of combination on Simple Support Functions(SSF), where S and C are viewed as single support functions on an agent's sincerity and competency respectively[17]. It can be easily proved that:

$$\tau_{multiply} \leq \tau_{min} \leq \tau_{ssf}$$

In our model an agent can choose different operators in different situations, or an agent can be designed with different characteristics in terms of how it evaluates others. Multiplication $\tau_{multiply}$ tends to be cautious but pessimistic, Bernoulli's combination shows optimistic behavior, while minimum operator is mild and a compromise between the two.

3.2 Degree of Deception and Degree of Sincerity

Given two agents, x and y, $s_x(y)$ denotes x's evaluation of y's sincerity on information that he delivered, and $0 \leq s_x(y) \leq 1$. The intended interpretation of this value is to what extent y is telling exactly what he believes in his own knowledge base. For example, y believes p at a degree of δ, but he may decide to tell x that he believes p at a degree of δ'. To allow agents to communicate both the content and the degree of beliefs, we assume the message takes the form: (p, δ_{yx}), where p is the information, δ_{yx} is the degree of belief delivered along with the information p, whilst the real degree held by agent y is δ_y. To minimize the negative effect of being deceived, agent x's task is to guess a value as close as possible to δ_y before making any changes that might be triggered by the information from y.

Using this general message format, we are able to model different levels of lies and deception. The degree of belief discussed in this paper is considered as a Baysian's probability measure, where $\delta(p) = 1$ means that p is accepted, $\delta(p) = 0$ means p is rejected or $\neg p$ is accepted and undetermined if $0 < p < 1$[6]. Particularly, $\delta(p) = \delta(\neg p) = \frac{1}{2}$ is considered as the way of expressing complete ignorance[4]. Therefore, in the extreme case, an agent can tell the opposite of what he believes (i.e. $\delta_y = 0$ but $\delta_{yx} = 1$ and vice versa) or tell something he is completely ignorant about ($\delta_y(p) = \delta_y(\neg p) = \frac{1}{2}$, but $0 < \delta_{yx} < 1$).

It is true in the real world, sometimes, truthful messages might be unbelievable, and sending them could be irrational. On the other hand, some lies might be believable, or at least could sway the hearer to reconsider or change its actions in favor of the sender. As $0 \leq \delta \leq 1$, there are boundaries as to what extent an agent can lie, and the degree of deception(γ) is thus defined:

$$\gamma = \frac{|\delta_{\mathrm{yx}} - \delta_{\mathrm{y}}|}{\max(\delta_{\mathrm{y}}, 1 - \delta_{\mathrm{y}})}$$

[4] Baysian theory is often criticized for its inappropriate representation of ignorance, but our definition on the degree of deception can be easily modified to fit into other definitions of degree of beliefs such as the one used in Dempster-Shafer's belief functions.

The sender agent's decision about the value of γ is affected by various factors, but mainly (and intuitively) by two: the utility and the believability. In other words, there is a optimal value for γ which can maximize his utility and also sounds true but not absurd. As this is not our major concern, we are not going to pursue this further here. If interested, please refer to Gmytra's[7] study on why and when it is feasible for an agent to lie to others.

The receiver agent x, on receiving a information couple (p, δ_{yx}), will have to guess the value of δ_y and use the obtained δ'_y along with the competency belief (c) for his further decision making. Assume agent x himself uses the above mechanism for γ to defraud others, then he is likely to make the assumption that y is using similar mechanisms for himself[4]. A reasonable guess can be achieved if we assume,

$$\text{degree of deception} \equiv 1 \text{ - degree of sincerity (i.e. } \gamma \equiv 1 - s)$$

Substituting γ with $1 - s$ in the above definition formula for γ, which is the subjective guess of γ, we have

$$\delta'_y = \frac{\delta_{\mathrm{yx}}}{1 \pm (1 - \mathrm{s})} \text{ and } \delta'_y \geq \frac{1}{2} \text{ where s} \neq 0 \tag{1}$$

OR

$$\delta'_y = \frac{\delta_{\mathrm{yx}} \pm (1 - \mathrm{s})}{1 \pm (1 - \mathrm{s})} \text{ and } \delta'_y < \frac{1}{2} \text{ where s} \neq 0 \tag{2}$$

In the case where $s = 1$, $\delta'_y = \delta_y$, which means x believes y is completely truthful in the information that y delivered. In the case of $s = 0$, agent x can just ignore the information and not make any changes to his knowledge base.

The "cleaned" degree of belief δ'_y can then be used together with the degree of competency c to obtain the degree of acceptance δ_x^y[5]. To obtain this value, the combination operators proposed in section 3.1 can be used. This will prepare a reliable input to belief change and knowledge base integration etc, which is illustrated by an example in section 5.

3.3 The Evaluation of Competency and Sincerity w.r.t. Topics

Sections 3.1 and 3.2 describe two ways of using degrees of competency (C) and degrees of sincerity (S) provided an agent already has the values c and s. But how and where does an agent get these values?

In the real world, we can have some general feelings about others' trustworthiness, such as x generally trusts y to a degree, which is denoted by $t_x(y)$ in section 3.1. But quite often, trust on a certain action is specific to task(s). Similarly, an information agent's trustworthiness specifically depends on the domain area of the incoming information. For example, we might trust our father's opinion on how to get our TV fixed if he is an an electronic engineer, but would not trust

[5] Degree of acceptance is the degree of belief at which an receiver agent will incorporate the information into his own knowledge base.

his opinion on the direction of our AI research. Therefore, a general static trust value is not sufficient and we need a context sensitive measure of trust. So we introduce a **topic** specific trust evaluation towards others in both competency and sincerity. Thus, we augment the notation with one more variable, the **topic** P, $c_x(y, P)$ is x's evaluation of y's competency with respect to P. Similarly we have $s_x(y, P)$.

According to whether there is any previous experience with a trustee, Marsh [16] suggests three possible situations that an agent might encounter in assessing other's trustworthiness. They are adapted to suit our needs and listed below,

1. The agent is not known, in this or similar topics.
2. The agent is known, but not in this or similar topics.
3. The agent is known and trusted in this or similar topics.

Inspired by Jonker and Trenur's trust dynamics[11], the above three situations can be handled respectively as shown below:

1. In the first case, without previous trust influencing experiences but with a given competency or sincerity threshold, c_δ or s_δ, the information source can be treated as either initially competitive $c_x(y, P) \geq c_\delta$ (faithful $s_x(y, P) \geq s_\delta$) or initially uncompetitive $c_x(y, P) < c_\delta$ (unfaithful $s_x(y, P) < s_\delta$). Practical strategies can be added to enable agents to choose different initial values in different situations.
2. If the agent is known in some other topics, the non-relevant previous experience can be used as a coefficient ($0 \leq \alpha \leq 1$) to adjust the initial competency (sincerity) as assigned in the first case.
3. For the third case, the previous experiences can be used to calculate the current competency (sincerity) using a *competency (sincerity) evaluation or update function*. Such functions can be defined to suit various applications by following the properties and constraints proposed by Jonker and Trenur[11]. Although such functions are applicable for both competency and sincerity evaluation, one may find that an agent's competency value is very unlikely to change as dramatically as the sincerity value does. Therefore, it is reasonable to expect a slow dynamic in competency evaluation/update functions as compared to the ones for sincerity.

The above procedures are applicable for information that contains a single topic. For information that contains more than one topic, we will have to obtain the competency value and the sincerity value for each individual topic first and then combine them using the same set of combination operators (i.e. $\tau_{multiply}$, τ_{min} and τ_{ssf}) proposed for trustworthiness evaluation in section 3.1.

Using formal methods to encode the incoming information, we can calculate the topics for it[15]. Topics of a sentence p are defined as the set of atoms occurring in p. For example, given $\phi = a \vee b$, $\phi' = a \wedge b$, $\psi = (a \to b)$, and $\psi' = (a \to \neg b)$, using uppercase letters A and B to denote the topic for proposition a and b respectively, we have

$$topic(\phi) = topic(\phi') = topic(\psi) = topic(\psi') = \{A, B\}$$

Consider the following example: Agent y told agent x that *ecommerce does not necessarily involve computer science* (i.e. a $\rightarrow \neg b$). There are two topics: ecommerce (A) and computer science (B) involved in the information delivered by y. Given the values of $c_x(y, a)$ and $c_x(y, b)$, to calculate the combined competency of agent y on this information, we have

$$c_x(y, \{A, B\}) = c_x(y, A) \oplus c_x(y, B)$$

The above function can easily be extended to calculate the combined competency (sincerity) for more than two topics.

4 Evaluating Trust on Passing-On Information Using Information Pedigree

It is known that a large majority of knowledge and beliefs come from indirect experiences, namely, communication. In the real world, communication is not necessarily restricted to conversation with other agents. It also includes emails, publications, books, newspapers and TV channels and more recently the Web. We do not distinguish these special information channels from agents as information sources. Instead, they are considered as special types of information sources and an agent must evaluate their trustworthiness.

Furthermore, the communicated information itself could come from the sender's indirect experiences as well. In such cases, the sender serves as a mediator who passes on information through from a source agent to a receiver agent.

Literature related to the trust of information sources often only consider the reciprocal trust attitudes among agents. There are few studies on the trust of passing-on information. Even in the real world, people tend to ignore the importance of passing-on information, which may result in the receiver's inability to diagnose the sources of conflict. Consequently, this will affect the receiver's evaluation of other agents, and as a result the quality of their acquired information.

For example, in a RoboCup scenario[6], the perception of a robot is limited, the robot who is far away from the ball has to rely on communication to obtain the estimated position of the ball. Suppose Robot-1 is close enough to perceive the ball position, he sends information about the position of the ball to the closest robot, say Robot-2, and Robot-2 passes this information onto Robot-3. Typically, Robot-3 will consider the received information as Robot-2's perception because the information source (Robot-1) is normally not mentioned. Suppose at the same time (the ball hasn't been moved yet), Robot-3 got a message stating a different position message from Robot-4 who is also close enough, then Robot-3 has to judge which position is more accurate before taking action.

In such situations where conflict occurs, it is crucial for an agent to identify the original sources that are responsible for the conflicting information. The need for this is intensified in highly risky situations.

[6] Thanks to James Brusey at RMIT University for the valuable discussion on RoboCup.

4.1 Representing Information Pedigrees

One natural and sensible approach a proactive rational agent could take to further clarify the conflicting information sources is to interrogate the sender agents. By conflict information sources we mean those agents whose information when considered together will spoil the consistency of the receiver agent's knowledge base.

To facilitate such interrogation, it is ideal that agents retain a reasonable depth of *information pedigree*, which records the origin of the information. It is widely accepted that agents receive information from two major channels via their perceptions directly from the environment and via communication with other agents. Thus, on other agents' query about the source of information an agent should have one of the following answers:

1. The information in doubt is my own observation.
2. The information in doubt is from agent 1 (maybe also from 2, 3 ... n), the pedigree tree is thus enlarged.

We depict different scenarios of information gathering using information pedigree diagrams in Fig. 1, where agents are nodes and the directed arcs stands for information route.

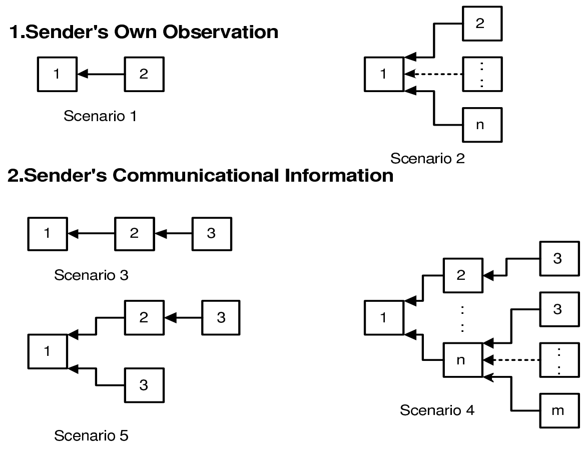


Fig. 1. Information Pedigrees

The trustworthiness assessment represented by Scenario 1 and 2 in the pedigree diagram concerns pairs formed by individual senders with the receiver. Scenario 1 illustrates that agent 2 tells agent 1 some of his observed information, whereas scenario 2 represents the situation where agent 1 receives observational information from multiple agents. With the degree of trustworthiness (or degree of sincerity and degree of competency) prepared as in section 3, scenario 1 turns into a belief change problem - how to maintain a consistent knowledge/belief base in face of new information. While scenario 2 becomes a fusion (or knowledge base merging) problem - how to incorporate new information from multiple sources.

Scenario 3, 4 and 5 illustrate more complicated cases where senders pass on information from the original sources to a receiver. The general information format (p, δ_{yx}) used in section 3.2 is only suitable for the pair-wise communication shown in scenario 1 and 2. Such representation is not sufficient for passing-on information, as the receiver agent also expects the sender agent to tell him some information about the original source. Therefore, we propose the following four tuple,

$$I_{yx} : (p, z, \delta_{yx}, \delta_{zyx})$$

where, x, y, z are agent IDs, p is the information, z is the original source of information p, δ_{yx} is agent y's degree of belief of p that agent y told agent x, δ_{zyx} is agent z's degree of belief of p that agent y told agent x. When $z = y$, that is, the sender is the original source, $(p, z, \delta_{yx}, \delta_{zyx})$ is then equivalent to (p, δ_{yx}).

Using the above information format, a sender can send a receiver not only his own degree of belief on p but also pass on other agent's degree of belief on p. Moreover, scenario 3, 4 and 5 can then be represented using information matrix similar to the following example for scenario 4.

Table 1. Agent 1's Information Matrix for Scenario 4

Information	Sender	Source	Sender's DOB*	Passing-on DOB*
p	2	3	δ_{21}	δ_{321}
...	...	...	...	...
p	n	3	δ_{n1}	δ_{3n1}
p	n	...	δ_{n1}	$\delta_{...n1}$
p	n	m	δ_{n1}	δ_{mn1}

∗DOB - Degree Of Belief

4.2 Information Pedigree Transformation Using Trust Evaluation

As we are talking about sincerity and competency, we have to consider the sender's trustworthiness on both the information itself and passing-on information about the original source.

Taking scenario 1 for instance, on receiving $(p, 3, \delta_{21}, \delta_{321})$, agent 1 faces the following uncertainties,

1) Agent 2 may/may not be faithful about his degree of belief on p, that is, $\delta_2 ? = \delta_{21}$, where δ_2 is agent 2's true degree of belief on p and δ_{21} is what agent 2 told agent 1. For evaluation, degree of sincerity on topic P (i.e. $s_1(2, P)$) could be employed here.
2) Agent 2 may/may not be competent in evaluating p. For evaluation, degree of competency on topic P (i.e. $c_1(2, P)$) could be used here.

3) Agent 2 may/may not be faithful about what agent 3 told him, that is, $\delta_{32}? = \delta_{321}$, where δ_{32} is the real value agent 2 received from agent 3 and δ_{321} is what agent 2 told agent 1 about δ_{32}. Using a similar approach as equation (1) and (2), we can estimate δ_{32}. The estimated value δ'_{32} is thus a function of δ_{321} and degree of sincerity $s_1(2,3)$ in the topic about agent 3.
4) Agent 2 may/may not be competent in passing information onto others, e.g. not certain whether the information is distorted during the process or not. This could be evaluated using the degree of competency $c_1(2,3)$ in the topic about agent 3.

Uncertainty of types 1) and 2) can be handled using the standard pair-wise trustworthiness evaluation proposed in section 3. For uncertainty types 3) and 4), the topic of the trustworthiness evaluation is no longer about p, but on another agent, which is why we have the notation $s_1(2,3)$ and $c_1(2,3)$ in the above list.

The ultimate goal of sincerity analysis is to get a close estimation of the source agent's degree of belief in p. In the case when agent 3 tells agent 1 directly about p, according to equation (1) and (2), we have,

$$\delta'_3 = f(\delta_{31}, s_1(3, P)) \tag{3}$$

But in the case of scenario 3 in fig. 1, agent 1 has to estimate the relationship between agent 2 and agent 3 and thus figure out the possibility that agent 3 lies to agent 2, which could be captured using the degree of sincerity $s_1(32, P)$. Therefore, using δ'_{32} and $s_1(32, P)$ to substitute δ_{31}, $s_1(3, P)$ in the above formula 3, agent 1 is able to estimate δ'_3 shown as follows,

$$\delta'_3 = f(\delta'_{32}, s_1(32, P))$$

To generalize the above process, we can use the following transformation shown in Fig. 2.

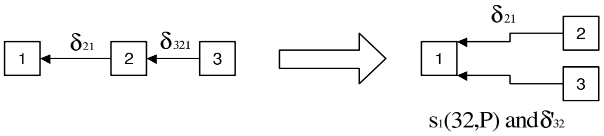


Fig. 2. Information Pedigree Transformation

Fig. 2 shows us that a 3-level pedigree tree can be transformed into a 2-level one. This will finally enable the use of belief change techniques, data fusion, merging or integration techniques to help the receiver agent maintain a consistent knowledge base.

As one may notice, the information pedigree diagrams in Fig. 1 are no more than 3-level in depth. This is because as soon as the receiver is notified about another agent by the sender, the receiver agent can directly contact the third involved agent for further information requested. Adding in another level of pedigree will unnecessarily increase the complexity of the trustworthiness evaluation process, which requires an agent to estimate the relationship of too many other agents. Therefore, to keep no more than one level of information source is sensible and sufficient for an agent to maintain the origin of communicational information.

5 An Example in Resolving Conflicting Information

Consider the following example,

John is interested in acquiring a property. After inspecting a potential property, he suspects that the dining room is a new extension and is not sure whether it is a legal extension. Then he inquired about this with the real estate agent, asked his lawyer to read the property's section 32 document, and also listened to two friends' opinions. Following is a summary of what he has gleaned from different sources:

Table 2. John's House Inspection

Agent	Information	
Lawyer: 1	No building approval in the past 7 years.	$e \rightarrow \neg l$
Real Estate Agent: 2	No building extension within the last 10 years.	$\neg e$
John's observation: 3	Dinning area seems to be newly extended	e
Friend A: 4	Statements about the purchaser's inspection.	s
Friend B: 5	Statements suggest a possible illegal extension.	$s \rightarrow \neg l$

Table 3. John's Social Knowledge

Agent	Information	Degree Of Belief δ_{yx}	Sincerity $s_x(y, P)$	Competency $s_x(y, P)$	Trustworthiness($\times$) $s_x(y, P)$
1	$e \rightarrow \neg l$	1	1	1	1
2	$\neg e$	0.9	0.4	0.8	0.32
3	e	1	1	0.5	0.5
4	s	1	1	0.8	0.8
5	$s \rightarrow \neg l$	0.9	1	0.6	0.6

In table 3, we assume that John has the values for degree of sincerity (s) and degree of competency (c) in his social knowledge base already. Then we can either work out the degree of trustworthiness (t) based on the values of s and c, or use s and c separately with δ_{yx} as shown in section 3.

The trustworthiness values in table 3 are obtained using operator $\tau_{multiply}$ introduced in section 3.1. This prepares the essential values that are required by Cantwell[1] in his process of resolving conflicting information.

Using the notation of relative support, Cantwell[1] obtains the order of sentences based on the joint trustworthiness[7] of the sources that have rejected the states. Taking the example illustrated in table 2 for instance, the conflict resolving process suggested by Cantwell can be described using the following steps:

1. Construct all the possible worlds based on the information state:
   $t_1 : e \to \neg l$; $t_2 : \neg e$; $t_3 : e$; $t_4 : s$; $t_5 : s \to \neg l$
   The possible worlds thus formed are
   $\{(e,l,s),(e,l,\neg s),(e,\neg l,s),(e,\neg l,\neg s),(\neg e,l,s),(\neg e,l,\neg s),(\neg e,\neg l,s),$ $(\neg e,\neg l,\neg s)\}$.
2. Given each agent(witness)'s individual trustworthiness, find out the joint trustworthiness of agents who reject the world, for example, for world (e,l,s), since agent 1 and 2 reject e, agent 1 and 5 reject l, agent 5 reject s, the joint trustworthiness of rejecting world (e,l,s) is then $t_1+t_2+t_1+t_5+t_5 = 3.52$. Similarly, by applying such procedure to each every possible world, we would thus associate a joint trustworthiness value to each world.
3. The worlds which have the largest joint trustworthiness value are the candidates to be rejected.

The above method does not guarantee a unique solution. Actually, in a special case when all the witnesses have equivalent trustworthiness, it generates the maximal consistent set of worlds, which is equivalent to the result of *theory extraction*[19]. Moreover, the worlds that are apt to be rejected are often the ones that do not satisfy the proposition that no agent is against. This implies that things uttered by agents take priority, if there is no objection, the receiver agent should accept it without doubt.

Alternatively, we can use equations (1) and (2) to work out the estimated degree of belief (δ'_y, $y = 1,2,3,4,5$) and combine δ'_y with c to obtain the degree of acceptance (δ^y_{john}, $y = 1,2,3,4,5$) that the receiver agent (John) is willing to consider. This will turn the above example into a standard belief revision problem, which could be easily solved using the existing belief revision techniques, such as Williams' adjustment[18].

After calculating δ^y_{john}, we have $\delta^1_{john} > \delta^4_{john} > \delta^5_{john} > \delta^3_{john} > \delta^2_{john}$, therefore,

$$\delta^1_{john}(1) : e \to \neg l$$

$$\delta^4_{john}(0.8) : s$$

$$\delta^5_{john}(0.54) : s \to \neg l$$

$$\delta^3_{john}(0.5) : e$$

$$\delta^2_{john}(0.45) : \neg e$$

As the real estate agent's claim has the lowest degree of acceptance, a rational agent will give up $\neg e$ and continue to accept that the extension is illegal to keep a consistent knowledge base.

[7] In this case, the joint trustworthiness of a set of agents is just the simple addition of the individual trustworthiness of each agent.

In addition, with the help of information pedigree, the receiver agent can first figure out the information that is in contradiction then ask the sender agents involved to further track down the information source. For example, after inquiring, agent 2 told the receiver agent, my information is from the city council, which is a definite trustworthy authority, the piece of information $\neg e$ from agent 2 will then take a much higher priority in the above ordering. In this case John's own observation e can be disgarded, i.e. he was wrong about the existence of an extension.

As shown by this example, the trust evaluation process proposed in this paper provides valuable insights into areas such as belief revision, conflict resolution, fusion and etc. In particular, the separation of sincerity and competency analysis offers a clarified view of the analysis required for evaluating the trustiworthiness of an information source, which in turn enables the analysis of passing-on information and offers a more flexible way of handling conflicting information. With information pedigree, we give agents another means for searching more evidence so as to arrive at a unique sensible solution. Furthermore, the trust evaluation based on experience makes the framework applicable in iterative situations as well.

6 Discussions and Future Work

Trust is often referred to as an adaption, a generalization or as a means for the reduction of complexity[16]. Adaption (via evolution) and generalization enables agents to learn from experience in beneficial ways and provides agents with means of reasoning about and to make general assumptions about other agents or their environment for which they are ignorant about. Therefore, trust evaluation is a valuable and essential survival skill for agents as they interact in an uncertain social and physical environment.

In this paper, we investigate a procedure for evaluating an agent's trustworthiness as an information source. By separating this process into competency analysis and sincerity analysis, we are able to deal with complicated cases of passing-on information, where the same information may reach the receiver agent through different routes. In order to keep information about the source agent, we introduce information pedigrees as a means to maintain the history of communicational information. Information sources' trustworthiness analysis and computation described herein can be used to prepare agents to perform fusion, weighted knowledge base merging, and multiple source conflict resolution.

We use the framework proposed in this paper as the basis for an agent-oriented implementation that supports the acquisition of information from multiple sources with different degree of trustworthiness. The system is under development using the Java-based multi-agent programming paradigm JADE together with Belief Revision and Theory Extraction engines[19] from SATEN[8].

[8] The Sagacious Agent for Theory Extraction and revisioN http://ebusiness.newcastle.edu.au/systems/saten.html

References

1. John Cantwell. Resolving conflicting information. *Journal of Logic, Language, and Information*, 7:191–220, 1998.
2. Cristiano Castelfranchi and Rino Falcone. Principles of trust for mas: Cognitive anatomy, social importance, and quantification. In Y Demazeau, editor, *the Third International Conference on Multi-Agent Systems*, pages 72–79, Los Alamitos, 1998. IEEE Computer Society.
3. R. Demolombe. To trust information sources: a proposal for a modal logical framework. In C. Castelfranchi and Y-H. Tan, editors, *the First International Workshop on Deception, Fraud and Trust in Agent Societies*, Minneapolis/St Paul, USA, 1998.
4. Bruce Edmonds. Modelling socially intelligent agents. *Applied Artificial Intelligence*, 12:677–699, 1998.
5. G Elofson. Developing trust with intelligent agents: An exploratory study. In C. Castelfranchi and Y-H. Tan, editors, *the First International Workshop on Deception, Fraud and Trust in Agent Societies*, pages 125–139, Minneapolis/St Paul, USA, 1998.
6. Peter Gärdenfors. *Knowledge in Flux - Modeling the Dynamics of Epistemic States.* The MIT Press, London, 1988.
7. Piotr J. Gmytrasiewicz and Edmund H. Durfee. Toward a theory of honesty and trust among communicating autonomous agents. *Group Decision and Negotiation*, 2:237–258, 1993.
8. John Hardwig. The role of trust in knowledge. *Journal of Philosophy*, 88:693–708, 1991.
9. N. R. Jennings. On being responsible. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-91)*, pages 93–102, Amsterdam, The Netherlands, 1992. Elsevier Science Publishers.
10. N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.
11. Catholijn M. Jonker and Jan Treur. Formal analysis of models for the dynamics of trust based on experiences. In Francisco J. Garijo and Magnus Boman, editors, *the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, volume 1647, pages 221–231, Berlin, 1999. Springer-Verlag: Heidelberg, Germany.
12. Wei Liu and Mary Anne Williams. A framework for multi-agent belief revision (part i: The role of ontology). In Norman Foo, editor, *12th Australian Joint Conference on Artificial Intelligence*, Lecture Notes in Artificial Intelligence, Sydney, Australia, 1999. Springer-Verlag.
13. Wei Liu and Mary Anne Williams. A framework for multi-agent belief revision (part ii: A layered model and shared knowledge structure). In *Electronic Transactions on Artificial Intelligence*, Uncertainty Frameworks in NMR2000, http://www.ida.liu.se/ext/etai/ufn/received/sframe.html, 2000.
14. Wei Liu and Mary Anne Williams. A framework for multi-agent belief revision. *Studia Logica*, 67 (2):291–312, 2001.
15. Pierre Marquis and Nadège Porquet. Decomposing propositional knowledge bases through topics. In *Partial Knowledge And Uncertainty: Independence, Conditioning, Inference*, Rome, Italy, 2000.

16. Stephen Paul Marsh. *Formalising Trust as a Computational Concept.* For the degree of doctor of philosophy, University of Stirling, 1994.
17. Glenn Shafer, editor. *A Mathematical Theory of Evidence.* Prinston University Press, New Jersey, 1976.
18. Mary Anne Williams. Transmutation of knowledge systems. In J. Doyle, E. Sandewall, and P. Torasso, editors, *4th International Conf. on Principles of Knowledge Representation and Reasoning*, pages 619 – 629. Morgan Kaufmann Publishers, 1994.
19. Mary Anne Williams and Aidan Sims. Saten: An object-oriented web-based revision and extraction engine. In *International Workshop on Nonmonotonic Reasoning (NMR'2000).*, Online Computer Science Abstract: http://arxiv.org/abs/cs.AI/0003059/, 2000.

Revisiting Asimov's First Law: A Response to the Call to Arms

David V. Pynadath and Milind Tambe

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
{pynadath,tambe}@isi.edu

Abstract. The deployment of autonomous agents in real applications promises great benefits, but it also risks potentially great harm to humans who interact with these agents. Indeed, in many applications, agent designers pursue adjustable autonomy (AA) to enable agents to harness human skills when faced with the inevitable difficulties in making autonomous decisions. There are two key shortcomings in current AA research. First, current AA techniques focus on individual agent-human interactions, making assumptions that break down in settings with teams of agents. Second, humans who interact with agents want guarantees of safety, possibly beyond the scope of the agent's initial conception of optimal AA. Our approach to AA integrates Markov Decision Processes (MDPs) that are applicable in team settings, with support for explicit safety constraints on agents' behaviors. We introduce four types of safety constraints that forbid or require certain agent behaviors. The paper then presents a novel algorithm that enforces obedience of such constraints by modifying standard MDP algorithms for generating optimal policies. We prove that the resulting algorithm is correct and present results from a real-world deployment.

1 Introduction

Agent applications, such as intelligent homes [6], smart organizations [9], and space missions [3], offer a vision of agent teams deployed to support critical human activities. While such deployment promises great benefits, it also runs a great risk of harm or cost to the humans involved. To mitigate this risk, researchers have pursued two general approaches. The first is inspired by Asimov's First Law of Robotics, "*A robot may not injure a human being or through inaction allow a human being to come to harm*" [1]. Here, researchers have emphasized tractable means of guaranteeing that agents obey *safety conditions* in planning and learning — unfortunately, despite the dramatic call to arms to ensure such safety [12], this approach enjoys only a small following.

Instead, a second, alternative approach of *adjustable autonomy* (AA) has gained more popularity. AA enables an agent to act with varying degrees of autonomy (e.g., in uncertain situations, the agent may transfer decision-making control to a more skilled, more knowledgeable human supervisor). Unfortunately, this research has, to date, focused on individual agent-human interactions. It has failed to address multi-agent team settings, such as our Electric Elves (henceforth E-Elves) application, where software assistants employ AA in helping human users coordinate with their colleagues. In particular, existing AA strategies can result in a rigid transfer of decision-making control to humans. In a team setting, such rigidity can cause costly miscoordination when the humans fail to make timely decisions.

This paper presents a marriage of both approaches, while addressing their key shortcomings in complex multiagent environments. For the foreseeable future, harnessing human skills via AA will be important when deploying agents in real-world settings. However, inaccuracies in an agent's world model can cause its seemingly optimal plans to violate safety, so additional guarantees are important. For instance, in E-Elves, modeling inaccuracies once caused a co-author's software assistant to autonomously cancel a critical meeting with his supervisor! Since these agents interact with different humans with varying preferences, the designers cannot provide them with a single model that represents everyone's safety requirements with complete accuracy. On the other hand, it is unreasonable (if not impossible) to have humans specify sufficient safety conditions to completely determine correct agent behavior. Thus, the synergy of AA reasoning and safety conditions is critical in multiagent domains.

This paper begins by describing our novel AA framework, which uses Markov Decision Processes (MDPs) [8] that encode individual and team costs and uncertainties. The use of MDPs allows the agents to compute optimal policies for flexible transfer of control and changes of coordination in AA, thus alleviating team miscoordination. The paper then integrates safety guarantees within the MDP framework. In particular, we enable humans to explicitly specify four different types of safety conditions: *forbidden actions*, *forbidden states*, *required actions*, and *required states*, realized as symbolic constraints on MDP states and actions, eliminating the need to determine sufficiently high-magnitude reward values to guarantee safety. We provide clear probabilistic semantics for these constraints, and present a novel algorithm that generates optimal MDP policies subject to these symbolic constraints. One key result presented is the correctness proof for the algorithm. By incorporating such explicit safety conditions, agents can guarantee safety despite potential MDP model inaccuracies that may arise. An additional advantage of using symbolic constraints is the support for constraint propagation as a preprocessing step, restricting the space of policies that standard value iteration must consider and thus providing significant gains in the efficiency of policy generation. Finally, while we present results of these algorithms as implemented in the context of MDPs for AA, they are applicable to MDPs in other domains as well.

2 Electric Elves

E-Elves [9] is a real-world experiment, where an agent team of 16 agents, including 12 proxies (assisting 12 people) have been running 24/7 for the past seven months at USC/ISI. Each proxy, called "Friday" (after Robinson Crusoe's servant, Friday), acts on behalf of its user in the agent team. Each Friday uses a teamwork model, STEAM [11] to manage its teamwork. If a user is delayed to a meeting, Friday can reschedule the meeting, relaying the delay to other Fridays, who in turn inform their human users of the delay. If there is a research presentation slot open, Friday responds to the invitation on behalf of its user. Friday can also order its user's meals and track the user's location.

Friday communicates through its user's workstation display and wireless devices (e.g., PDAs, mobile phones).

AA is critical in Friday agents. The more autonomous Friday is, the more effort it saves its user. However, in our early implementation, Friday sometimes made costly mistakes when acting autonomously (e.g., unwarranted meeting cancellation). Such mistakes arose, in part, because of the uncertainty in the domain and the real-world costs of actions. Given such costs and uncertainties, AA aims to enable each Friday to intelligently decide between acting autonomously and transferring control to a human user.

Unfortunately, the team context of domains like E-Elves raises a novel AA challenge in the potential for miscoordination when transferring control to human users. In another example from our early implementation, Friday agents would sometimes rigidly transfer control to their users (e.g., to ask whether they planned to attend a meeting). In some cases, a user would be unable to respond, sometimes due to getting stuck in traffic. While waiting for a response, Friday's inaction would cause miscoordination with its teammates (other Friday agents), since it failed to inform them whether its user would attend. This, in turn meant that the other attendees (humans) wasted their time waiting for their teammate. Later, under a different set of circumstances, Friday did not ask the user and instead, following its best guess, autonomously informed the other Fridays that its user would not attend the meeting. However, the user *did* plan to attend, so the human team suffered a serious cost from receiving this incorrect information from Friday. Friday must instead plan a course of action that makes the best tradeoff possible between the possible costs, both present and future, of inaction and of incorrect action.

3 General Approach to AA in Teams

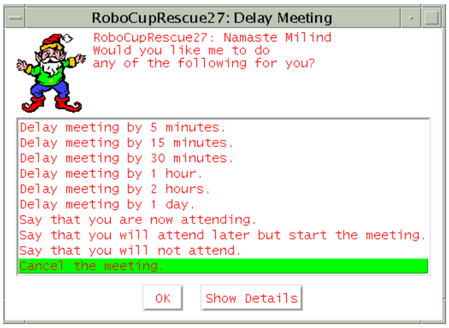
We consider the general problem of AA in teams as follows. We assume a team with a joint activity, α, and a human team member who performs role, ρ, in α. A software assistant must aid the user in performing ρ (e.g., by securing more resources from the team) and must keep the team informed about whether the user will perform ρ. Given the uncertainty in whether its user can perform ρ, the AA challenge for the software assistant is to decide when to act autonomously and when to transfer decision-making control to the user (e.g., when informing the team that the user cannot perform ρ). For instance, in E-Elves, α is a meeting, ρ is the user's attendance at the meeting, and the key resource is time. Friday attempts to ensure that its user can attend the meeting. One difficulty here is that rigid transfer-of-control regimes cause team miscoordination (as discussed in Section 2). Another difficulty is balancing the user's needs in performing ρ against the team's needs for α to succeed.

We provide a general approach to this AA problem in teams, based on Markov Decision Processes (MDPs) [8]. In this paper, we use reward functions over both state-action pairs ($R_A : S \times A \to [0, 1]$) and states ($R_S : S \to [0, 1]$). The key elements of our approach are as follows. First, the MDP reward functions factor in the individual benefits due to performing ρ, team costs due to changes in α, costs of consulting a user, etc. Second, an agent does not rigidly transfer decision-making control to a user; instead, it defines an MDP policy for flexibly transferring control back and forth to ensure team coordination. Thus, if an agent transfers control to a user and the user fails to respond, then the MDP policy guides the agent to act autonomously, to avoid team miscoordination. Third, when an agent is required to act autonomously to avoid miscoordination, the MDP policy mitigates the significant risks by guiding the agent to change its coordination arrangements, postponing or reordering activities to "buy time" to lower decision cost/uncertainty. For instance, in E-Elves, an agent may delay a meeting, improving the user's chances of attending the meeting, and also buying more time for user input.

To generate such an MDP, the set of actions, A, includes: wait (do nothing), reduce autonomy (ask the user for information), autonomously inform other agents that the user is performing (or not performing) ρ, or change the coordination of α (e.g., asking for delays in performing ρ). Each state, $s \in S$, is a tuple: $\{$*rel-time-*ρ, *user-status*, α*-status*, *user-response*, other α attributes$\}$. Here, *rel-time-*ρ measures time left until performance of ρ in α. Similarly, *user-status* enables an agent to (probabilistically) estimate the time the user will actually need before performing ρ in α.

One example of such an AA MDP is the *delay MDP*, covering all meetings for which Friday may act on behalf of its user. Friday can take any of 12 actions, specializations of the general actions already described. For example, informing other agents that the user is performing ρ corresponds to announcing that the user will attend a meeting. Changing the coordination of α corresponds to delaying the meeting. When Friday reduces its autonomy and asks, its user can respond with any of the 10 options from Figure 1a. In the delay MDP's state representation, *rel-time-*ρ becomes *rel-time-meeting*, *user-status* becomes *user location*, α*-status* becomes *status of meeting*, etc. Figure 1b shows a portion of the state space, showing the *rel-time-meeting*, *user-response*, and *user location* features. The figure also shows some state transitions (a transition labeled "delay n" corresponds to the action "delay by n minutes"). Each state contains other features (e.g., *previous-delays*), not pictured, relevant to the overall joint activity, for a total of 2760 states in the MDP for each individual meeting.

In general, our AA MDP framework uses a reward function $R(s, a) = f($*rel-time-*$\rho(s)$, *user-status*(s), α*-status*$(s), a)$, where the first three components reflect the value of the team activity in its current status (e.g., active but without user performing ρ), while the


RoboCupRescue27: Delay Meeting

RoboCupRescue27: Namaste Milind
Would you like me to do
any of the following for you?

Delay meeting by 5 minutes.
Delay meeting by 15 minutes.
Delay meeting by 30 minutes.
Delay meeting by 1 hour.
Delay meeting by 2 hours.
Delay meeting by 1 day.
Say that you are now attending.
Say that you will attend later but start the meeting.
Say that you will not attend.
Cancel the meeting.

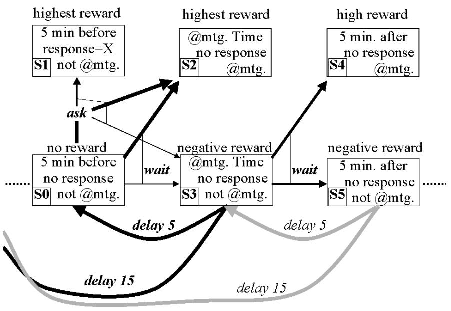


Fig. 1. (a) Dialog box for delaying meetings in E-Elves.(b) A small portion of the *delay MDP*.

action component reflects the cost of changing the coordination of the team activity, α. This reward function has a maximum when the user performs ρ (e.g., the delay MDP encodes a maximum reward in the state where the user is at the meeting location when the meeting starts). The *delay MDP* divides this function further into more specific components. One such component, denoted r_{user}, focuses on the user attending the meeting at the meeting time. r_{user} gives the agent incentive to delay meetings when its user's late arrival is possible, but large delays incur a team cost from rearranging schedules. The team cost is considered by incorporating a negative reward, denoted r_{repair}, with magnitude proportional to the number of delays so far and the number of attendees, into the delay reward function. However, without a delay, the other attendees may waste time waiting for the agent's user to arrive. Therefore, the reward function includes a component, r_{time}, that is negative in states after the start of the meeting if the user is absent, but positive otherwise. The overall reward function for a state, s, is a weighted sum: $r(s) = \lambda_{user}r_{user}(s) + \lambda_{repair}r_{repair}(s) + \lambda_{time}r_{time}(s)$.

The delay MDP's transition probabilities represent the likelihood that a user movement (e.g., from office to meeting location) will occur in a given time interval. Figure 1b shows multiple transitions due to "ask" and "wait" actions, with the thickness of the arrows reflecting their relative probability. The designer encodes the initial probabilities, which a learning algorithm may then customize. Other state transitions correspond to uncertainty associated with a user's response (e.g., when the agent performs the "ask" action, the user may respond with specific information or may not respond at all, leaving the agent to effectively "wait").

Given the MDP's state space, actions, transition probabilities, and reward function, an agent can use *value iteration* to generate a policy $P{:}S{\rightarrow}A$ that specifies the optimal action in each state [8]. These policies allow user-assistant agents to avoid rigidly committing to transfer-of-control decisions. Thus, if the agent decides to give up autonomy, it does not wait indefinitely for a user response. Instead, the agent continuously reassesses the developing situation, possibly changing its previous autonomy decisions. For instance, one possible policy, for a subclass of meetings, specifies "ask" in state **S0** of Figure 1b (i.e., the agent gives up some autonomy). If the world reaches state **S3**, the policy specifies "wait". However, if the agent then reaches state **S5**, the policy chooses "delay 15", which the agent then executes autonomously. Thus, the agent's AA is an ongoing process, rather than a single decision.

In addition, when an agent must act autonomously to avoid miscoordination, it can minimize the risk of error by carefully planning a change in coordination (e.g., delaying the meeting). The delay MDP is especially suitable for producing such a plan, because it generates policies by looking ahead at the future costs of team miscoordination and erroneous actions. For instance, through the delay MDP, an agent can reason that a short delay will buy more time for a user to respond to its query, incurring a small communication cost but potentially reducing the uncertainty of a costly decision. Thus, before deciding to autonomously cancel a meeting, an agent might choose a 15-minute delay to give time for an absent user to arrive, or at least respond.

4 Avoiding Harm: Safety Constraints

The MDP-generated policies ensure that an agent's actions are optimal with respect to its current model. However, they do not automatically ensure that agents will avoid harm and obey the *safety constraints* of interest to a human interacting with it. The MDP model of costs and likelihoods may contain inaccuracies, perhaps from the agent's changing environment. Furthermore, in domains such as ours, where agents must interact with different people, the costs and likelihoods may vary significantly from person to person. As an example, students may want to enforce a safety constraint that their Fridays never cancel a meeting with a faculty member, but faculty may not wish to enforce the same (a purely hypothetical case). An agent could potentially learn an individualized MDP model with enough user feedback. However, learning may require an arbitrarily large number of interactions with the humans involved. In the meantime, the agent's incorrect model can still lead to catastrophic results.

4.1 Definition of Constraints

To prevent such catastrophes, it is important to allow humans to directly specify important safety conditions. One way to ensure that agents avoid harm is to forbid them from entering specific states or performing specific actions in specific states. For instance, a user may forbid an agent from autonomously cancelling a meeting in some states (e.g., any state involving a meeting with a supervisor). We define such **forbidden-action** constraints to be a set, C_{fa}, where each element constraint is a boolean function, $c_{fa} : S \times A \rightarrow \{t, f\}$. Similarly, we define **forbidden-state** constraints to be a set, C_{fs}, with elements, $c_{fs} : S \rightarrow \{t, f\}$. If a constraint returns t for a particular domain element (either state or state-action pair, as appropriate), then the constraint applies to the given element. For example, a forbidden-action constraint, c_{fa}, forbids the action a from being performed in state s if and only if $c_{fa}(s, a) = t$.

To provide probabilistic semantics, suitable for an MDP context, we first provide some notation. Denote the probability that the agent will ever arrive in state s_f after following a policy, P, from an initial state s_i as $\Pr(s_i \overset{*}{\rightarrow} s_f | P)$. Then, we define the semantics of a forbidden-state constraint c_{fs} as requiring $\Pr(s_i \overset{*}{\rightarrow} s_f | P) = 0$. The semantics given to a forbidden-action constraint, c_{fa}, is a bit more complex, requiring $\Pr(s_i \overset{*}{\rightarrow} s_f \wedge P(s_f) = a | P) = 0$ (i.e., c_{fa} forbids the agent from entering state s_f *and* then performing action a). In some cases, an aggregation of constraints may forbid *all* actions in state s_f. In this case, the conjunction allows the agent to still satisfy all forbidden-action constraints by avoiding s_f (i.e., the state s_f itself becomes forbidden). Once a state, s_f, becomes indirectly forbidden in this fashion, any action that potentially leads the agent from an ancestor state into s_f likewise becomes forbidden. Hence, the effect of forbidding constraints can propagate backward through the state space, affecting state/action pairs beyond those which cause immediate violations.

The forbidding constraints are powerful enough to express a variety of safety constraints. For instance, some E-Elves users have forbidden their agents from rescheduling meetings to lunch time. To do so, the users provide a feature specification of the states they want to forbid, such as *meeting-time*=12 PM. Such a specification generates a forbidden-state constraint, c_{fs}, that is true in any state, s, where *meeting-time*=12 PM in s. Similarly, some users have forbidden cancellations by providing a specification of the action they want to forbid, *action*="cancel". This generates a forbidden-action constraint, c_{fa}, that is true for any state/action pair, (s, a), with $a =$"cancel". Users can easily create more complicated constraints by specifying values for multiple features, as well as by using comparison functions other than $=$ (e.g., $\neq$, $>$). We have also noticed synergistic interactions between AA and forbidden constraints, such as forbidding an agent from taking autonomous action in some states (i.e., the agent must instead reduce its level of autonomy and ask the user for input).

While powerful, the forbidden constraints are unable to express all safety constraints of interest. In particular, some safety constraints *require* that an agent to visit certain states states or to perform a specific action in certain states. For instance, in our E-Elves domain, we may require our agents to, at some point during their execution, notify our teammates about our state, whether *attending*, *not attending*, or *cancelled*, as we want to avoid team miscoordination through complete inaction. We do not require this notification property to be true in any particular state, or in all states. The intent is only to ensure that the teammates are eventually notified, so it is difficult to express this property as a forbidden-state constraint (e.g., forbidding all states without this notification property).

Hence, we introduce **required-state** and **required-action** constraints, defined as sets, C_{rs} and C_{ra}, respectively, with elements analogous to their forbidding counterparts. The interpretation provided to the required-state constraint is symmetric, but opposite to that of the forbidden-state constraint: $\Pr(s_i \overset{*}{\rightarrow} s_f | P) = 1$. Thus, from any state, the agent *must* eventually reach a required state, s_f. Similarly, for the required-action constraint, $\Pr(s_i \overset{*}{\rightarrow} s_f \wedge P(s_f)=a | P) = 1$. The users specify such constraints as they do for their forbidding counterparts (i.e., by specifying the values of the relevant state features or action, as appropriate). In addition, the requiring constraints also propagate backward. Informally, the forbidden constraints focus locally on specific states or actions, while the required constraints express global properties over all states.

4.2 Value Iteration with Constraint Propagation

We have extended standard value iteration to also consider constraint satisfaction when generating optimal policies. The value of each state is no longer a single value, but a tuple $\langle F, N, U \rangle$, where F is a boolean indicating whether the state is forbidden or not, N is a set of requiring constraints that will be satisfied, and U is the expected value (as in traditional value iteration),. For instance, if the value of a state, $V(s) = \langle t, \{c_{rs}\}, 0.3 \rangle$, then executing the policy from state s will achieve an expected value of 0.3 and will satisfy required-state constraint c_{rs}. However, it is not guaranteed to satisfy any other required-state, nor any required-action, constraints. In addition, s is forbidden, so there is a nonzero probability of violating a forbidden-action or forbidden-state constraint. We do not record *which* forbidding constraints the policy violates, since violating any one of them is equally bad. We *do* have to record which requiring constraints the policy satisfies, since satisfying all such constraints is preferable to satisfying only some of them. Therefore, the size of the value function grows linearly with the number of requiring constraints, but is independent of the number of forbidding constraints.

We initialize the value function over states as follows:

$$V^0(s) \leftarrow \langle \bigvee_{c \in C_{fs}} c(s), \{c \in C_{rs} | c(s)\}, R_S(s)\rangle \tag{1}$$

Thus, s is forbidden if any forbidden-state constraints apply and is needed by any required-state constraints that apply.

In value iteration, we must define an updated value function V^{t+1} as a refinement of the previous iteration's value function, V^t. States become forbidden in V^{t+1} if they violate any constraints directly or if *any* of their successors are forbidden according to V^t. States satisfy requirements if they satisfy them directly or if *all* of their successors satisfy the requirement. We define S' to be the set of all successors: $\{s' \in S | M^a_{ss'} > 0\}$. The following expression provides the precise definition of this iterative step:

$$\begin{aligned} V^{t+1}(s) \leftarrow \max_{a \in A} \Bigg\langle & \bigvee_{c \in C_{fs}} c(s) \vee \bigvee_{c \in C_{fa}} c(s,a) \vee \bigvee_{V^t(s')=\langle F',N',U'\rangle, s' \in S'} F', \\ & \{c \in C_{rs} | c(s)\} \cup \{c \in C_{ra} | c(s,a)\} \cup \bigcap_{V^t(s')=\langle F',N',U'\rangle, s' \in S'} N', \\ & R_S(s) + R(s,a) + \sum_{V^t(s')=\langle F',N',U'\rangle, s' \in S'} M^a_{ss'} U' \Bigg\rangle \end{aligned} \tag{2}$$

The maximization uses the following preference ordering, where $x \prec y$ means that y is preferable to x:

$$\begin{aligned} \langle t, N, U\rangle &\prec \langle f, N', U'\rangle \\ \langle F, N, U\rangle &\prec \langle F, N' \supset N, U'\rangle \\ \langle F, N, U\rangle &\prec \langle F, N, U' > U\rangle \end{aligned}$$

In other words, satisfying a forbidden constraint takes highest priority, satisfying more requiring constraints is second, and increasing expected value is last. We define the optimal action, $P(s)$, as the action, a, for which the final $V(s)$ expression above is maximized.

Despite the various set operations in Equation 2, the time complexity of this iteration step exceeds that of standard value iteration by only a linear factor, namely the number of constraints, $|C_{fs}| + |C_{fa}| + |C_{rs}| + |C_{ra}|$. The efficiency derives from the fact that the constraints are satisfied/violated independently of each other. The determination of whether a single constraint is satisfied/violated requires no more time than that of standard value iteration, hence the overall linear increase in time complexity.

Because expected value has the lowest priority, we can separate the iterative step of Equation 2 into two phases: constraint propagation and value iteration. During the constraint-propagation phase, we compute only the first two components of our value function, $\langle F, N, \cdot\rangle$. The value-iteration phase computes the third component, $\langle \cdot, \cdot, U\rangle$, as in standard value iteration. However, we can ignore any state/action pairs that, according to the results of constraint propagation, violate a forbidding constraint ($\langle t, N, \cdot\rangle$) or requiring constraint ($\langle f, N \subset C_{rs} \cup C_{ra}, \cdot\rangle$). Because of the componentwise independence of Equation 2, the two-phase algorithm computes an identical value function as the original, single-phase version (over state/action pairs that satisfy all constraints).

The worst-case time complexity of the two versions is also identical. However, in practice, the two-phase algorithm achieves significant speedup through the rapid elimination of state/action pairs during constraint propagation. While the single-phase version generally requires as much time as standard value iteration (i.e., without constraints), the two-phase algorithm often computes a policy in dramatically *less* time, as the results in Section 5 demonstrate.

4.3 Correctness of Propagation Algorithm

Given a policy, P, constructed according to the algorithm of Section 4.2, we must show that an agent following P will obey the constraints specified by the user. If the agent begins in some state, $s \in S$, we must prove that it will satisfy all of its constraints if and only if $V(s) = \langle f, C_{ra} \cup C_{rs}, U \rangle$. We prove the results for forbidding and requiring constraints separately.

Theorem 1. *An agent following policy, P, with value function, V, generated as in Section 4.2, from any state $s \in S$ (for a finite set S) will violate a forbidding constraint with probability zero if and only if $V(s) = \langle f, N, U \rangle$ (for some U and N).*

Proof: We first partition the state space, S, into subsets, S_k, defined to contain all states that can violate a forbidding constraint after a minimum of k state transitions. In other words, S_0 contains those states that violate a forbidding constraint directly; S_1 contains those states that do not violate any forbidding constraints themselves, but have a successor state (following P) that does (i.e., a successor state in S_0); S_2 contains those states that do not violate any forbidding constraints, nor have any successors that do, but who have at least one successor state that has a successor state that does (i.e., a successor state in S_1); etc. There are at most $|S|$ nonempty subsets in this mutually exclusive sequence (in which case, each subset contains exactly one state from S). To make this partition exhaustive, the special subset, S_{never}, contains all states from which the agent will never violate a forbidding constraint by following P. We first show, by induction over k, that $\forall s \in S_k$ $(0 \leq k \leq |S|)$, $V(s) = \langle t, N, U \rangle$, as required by the theorem.

Basis step (S_0): By definition, the agent will violate a forbidding constraint in $s \in S_0$. Therefore, either $\exists c \in C_{fs}$ such that $c(s) = t$ or $\exists c \in C_{fa}$ such that $c(s, P(s)) = t$, so we know, from Equation 2, $V(s) = \langle t, N, U \rangle$.

Inductive step ($S_k, 1 \leq k \leq |S|$): Assume, as the induction hypothesis, that $\forall s' \in S_{k-1}$, $V(s') = \langle t, N', U' \rangle$. By the definition of S_k, each state, $s \in S_k$, has at least one successor state, $s' \in S_{k-1}$. Then, according to Equation 2, $V(s) = \langle t, N, U \rangle$, because the disjunction over S' must include s', for which $F' = t$.

Therefore, by induction, we know that for all $s \in S_k$ $(0 \leq k \leq |S|)$, $V(s) = \langle t, N, U \rangle$. We now show that $\forall s \in S_{\text{never}}$, $V(s) = \langle f, N, U \rangle$. We prove, by induction over t, that, for any state, $s \in S_{\text{never}}$, $V^t(s) = \langle f, N, U \rangle$.

Basis step (V^0): By definition, if $s \in S_{\text{never}}$, there cannot exist any $c \in C_{fs}$ such that $c(s) = t$. Then, from Equation 1, $V^0(s) = \langle f, N^0, U^0 \rangle$.

Inductive step ($V^t, t > 0$): Assume, as the inductive hypothesis, that, for any $s' \in S_{\text{never}}$, $V^{t-1}(s') = \langle f, N', U' \rangle$. We know that $V^t(s) = \langle f, N^t, U^t \rangle$ if and only if all three disjunctions in Equation 2 are false. The first is false, as described in the basis step. The second term is similarly false, since, by the definition of S_{never}, there cannot exist any $c \in C_{fa}$ such that $c(s, P(s)) = t$. In evaluating the third term, we first note that $S' \subseteq S_{\text{never}}$. In other words, all of the successor states of s are also in S_{never} (if successor $s' \in S_k$ for some finite k, then $s \in S_{k+1}$). Since all of the successors are in S_{never}, we know, by the inductive hypothesis, that the disjunction over V^{t-1} in all these successors is false. Therefore, all three disjunctive terms in Equation 2 are false, so $V^t(s) = \langle f, N^t, U^t \rangle$.

Therefore, by induction, we know that for all $s \in S_{\text{never}}$, $V(s) = \langle f, N, U \rangle$. By the definition of the state partition, these two results prove the theorem as required. □

Theorem 2. *An agent following policy, P, with value function, V, generated as described in Section 4.2, from any state $s \in S$ (for a finite set S) will satisfy each and every requiring constraint with probability one if and only if $V(s) = \langle F, C_{ra} \cup C_{rs}, U \rangle$ (for some U and F).*

Proof Sketch: The proof parallels that of Theorem 1, but with a state partition, S_k, where k corresponds to the *maximum* number of transitions before satisfying a requiring constraint. However, here, states in S_{never} are those that *violate* the constraint, rather than satisfy it. Some cycles in the state space can prevent a guarantee of satisfying a requiring constraint within any fixed number of transitions, although the probability of satisfaction *in the limit* may be 1. In our current constraint semantics, we have decided that such a situation fails to satisfy the constraint, and our algorithm behaves accordingly. Such cycles have no effect on the handling of forbidding constraints, where, as we saw for Theorem 1, we need consider only the *minimum*-length trajectory. □

The proofs of the two theorems operate independently, so the policy-specified action will satisfy all constraints, if such an action exists. The precedence of forbidding constraints over requiring ones has no effect on the optimal action in such states. However, if there are conflicting forbidding and requiring constraints in a state, then the preference ordering causes the agent to choose a policy that satisfies the forbidding constraint and violates a requiring constraint. The agent can make the opposite choice if we simply change the preference ordering from Section 4.2. Regardless of the choice, from Theorems 1 and 2, the agent can use the value function, V, to identify the existence of any such violation and notify the user of the violation and possible constraint conflict.

5 Evaluation

We begin with an evaluation of the overall utility of E-Elves and then evaluate the MDP-based AA framework and the safety constraints separately. We have used the E-Elves system (described in Section 2) within our research group, and here are some key facts about our experience so far:

- Running since 6/1/2000, 24 hours/day, 7 days/week (occasionally interrupted for enhancements).
- Mean number of messages/day among agents: 85
- Number of meetings: 689; autonomous rescheduling: 213; user rescheduling: 152

– Number of meeting presenters selected: 10, 8 autonomously and 2 by users
– Number of meals ordered: 54, with mean 1.4 people per order

The fact that E-Elves users were (and still are) willing to use the system over such a long period and in a capacity so critical to their daily lives is a testament to its effectiveness.

Our MDP-based approach to AA has provided much value to the E-Elves users, as attested to by the large number (689) of meetings that the users have allowed their agent proxies to monitor using the delay MDP. The large number (213) of autonomous rescheduling corresponds to a large amount of user effort saved. As for the correctness of the MDP approach, over the entire span of their operation, the agents have performed without any catastrophic mistakes. For example, the policy described in Section 3 avoided the problems with rigid transfer of control, as described in Section 2. When giving up autonomy, the agent's policy specified waiting a certain amount of time to give the user a chance to respond. However, as the meeting drew closer, the policy eventually preferred taking back autonomy, so that the agent can act on its own before any miscoordination takes place. Figure 2 demonstrates how often the agents required this ability. The solid line plots the percentage of meetings vs. the number of transfers of autonomy. A transfer of autonomy takes place either when the agent asks the user for input or else when the agent, having asked the user for input without receiving it, stops waiting and acts autonomously. For some simple meeting instances, the agent was able to act completely autonomously, without ever giving up control (e.g., the user arrives at the meeting early). Ignoring these "easy" cases, the dashed line shows an alternate histogram over only those meetings for which the agent had to ask the user at least once. Although the current agents do occasionally make mistakes, these errors are typically on the order of asking for input a few minutes earlier than desired, etc. Unfortunately, the inherent subjectivity and intrusiveness of user feedback has complicated our determination of the optimality of the agents' decisions.

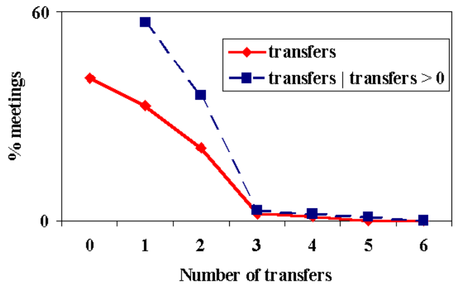


Fig. 2. Histogram of transfers of control per meeting.

In evaluating our safety-constraint framework, we have already proven the correctness of our algorithm. The constraints have also been useful. For instance, one typical user has added five safety constraints. Two forbidden-state constraints prevent meetings on Saturday or Sunday (e.g., forbid *meeting-day*=Sunday). A second forbidden-state constraint prevents his agent from rescheduling a meeting to lunch time *if* not originally scheduled at that time (i.e., forbid combination of *meeting-time*=12 PM and *previous-delays*$>$ 0). The user has also specified a forbidden-action constraint preventing cancellations in meetings with his superiors (i.e., forbid *action*="cancel"). Therefore, his agent never performs any erroneous, autonomous cancellations.

To evaluate the synergy between AA and constraints, we merged user-specified constraints from all E-Elves users, resulting in a set of 10 distinct constraints. We started with an unconstrained instance of the *delay MDP* and added these constraints one at a time, counting the policies that satisfied the applied constraints. We then repeated these experiments on expanded instances of the *delay MDP*, where we increased the initial state space by increasing the frequency of decisions (i.e., adding values to the *rel-time-meeting* feature). Figure 3a displays these results (on a logarithmic scale), where line A corresponds to the original *delay MDP* (2760 states), and lines B (3320 states), C (3880 states), and D (4400 states) correspond to the expanded instances. Each data point is a mean over five different orderings of constraint addition. For all four MDPs, the constraints substantially reduce the space of possible agent behaviors. For instance, in the original *delay MDP*, applying all 10 constraints eliminated 1180 of the 2760 original states from consideration, and reduced the mean number of viable actions per acceptable state from 3.289 to 2.476. The end result is a 50% reduction in the size ($\log_{10}$) of the policy space. On the other hand, constraints alone did not provide a complete policy, since all of the plots stay well above 0, even with all 10 constraints. Since none of the individual users were able/willing to provide 10 constraints, we cannot expect anyone to add enough constraints to completely specify an entire policy. Thus, value iteration is still far from redundant.

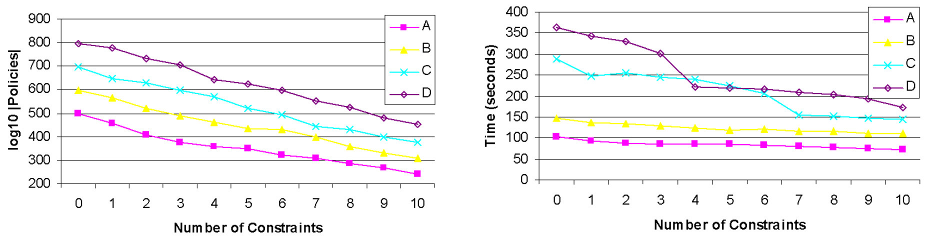


Fig. 3. (a) Number of possible policies (logarithmic). (b) Time required for policy generation.

The constraints' elimination of behaviors also decreases the time required for policy generation. Figure 3b plots the total time for constraint propagation and value iteration over the same four MDPs as in Figure 3a (averaged over the same five constraint orderings). To smooth out any variability in machine load, each data point is also a mean over five separate iterations, for a total of 25 iterations per data point. The values for the zero-constraint case correspond to standard value iteration without constraints. The savings in value iteration over the restricted policy space dramatically outweigh the cost of prepropagating the additional constraints. In addition, the savings increase with the size of the MDP. For the original *delay MDP* (A), there is a 28% reduction in policy-generation time, while for the largest MDP (D), there is a 53% reduction. Thus, the introduction of constraints speeds up policy generation, providing another example of the synergy between the two components of our implementation. We are continuing to conduct experiments to more exhaustively cover the set of possible constraints, constraint orderings, state space sizes, etc.

6 Summary and Related Work

It is a tribute to Asimov's creative intellect that his laws of robotics have not only fascinated generations of readers, but also inspired a research direction in autonomous agents. This paper builds on several key themes from prior work in this direction, as well as previous AA research. Our contribution to this line of research is to explicitly address safety in the context of AA, as well as to address AA in team settings (where agents must simultaneously satisfy team and individual responsibilities). This paper also introduces safety constraints that forbid or require certain states or actions in the context of MDPs. This paper presents an algorithm to propagate such constraints, addressing constraint interactions in optimal policy generation for the MDP and proving its correctness. It also presents experimental results that demonstrate the utility, in both safety and efficiency, of this algorithm in a real-world domain.

Existing research has already addressed the Asimovian concept of safety by emphasizing computationally tractable means of avoiding harm within the context of classical planning [12] and finite-state machines [5]. In other related work, previous AA research, in user control of robots [3] or in mixed-initiative planning [4], has focused on interactions between an individual agent and an individual human. Our research builds on both traditions, contributing the use of MDPs for AA in agent team settings and then providing a novel extension of supporting safety constraints. Our forbidden-action constraints support human-specified partial policies, as developed by other researchers [2,7]. However, constraints over states, as well as the required-action constraints, raise the novel issue of constraint propagation not addressed earlier. In previous work [10], we presented a specialized version of our MDP approach to AA, that focused on the delay MDP. This paper contributes a generalization of that approach using notions of role, ρ, and team activity, α and integrates four types of safety constraints absent in that work. The success we have achieved by combining and extending these two approaches (safety conditions and AA) within E-Elves demonstrates the value of such an approach in such real-world, multiagent domains.

References

1. Isaac Asimov. Runaround. *Astounding Science Fiction*, pages 94–103, 1942.
2. Craig Boutilier, Ray Reiter, Mikhail Soutchanksi, and Sebastian Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the National Conference on Artificial Intelligence*, pages 355–362, 2000.
3. Gregory A. Dorais, R. Peter Bonasso, David Kortenkamp, Barney Pell, and Debra Schreckenghost. Adjustable autonomy for human-centered autonomous systems on mars. In *Proceedings of the International Conference of the Mars Society*, 1998.

4. George Ferguson, James Allen, and Brad Miller. TRAINS-95 : towards a mixed initiative planning assistant. In *Proceedings of the Conference on Artificial Intelligence Planning Systems*, pages 70–77, May 1996.
5. Diana F. Gordon. Asimovian adaptive agents. *Journal of Artificial Intelligence Research*, 13:95–153, 2000.
6. Victor Lesser, Michael Atighetchi, Brett Benyo, Bryan Horling, Anita Raja, Regis Vincent, Thomas Wagner, P. Xuan, and S. Zhang. The UMASS intelligent home project. In *Proceedings of the Conference on Autonomous Agents*, pages 291–298, Seattle, USA, 1999.
7. Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Proceedings of Advances in Neural Information Processing Systems*, pages 53–79, 1997.
8. Martin L. Puterman. *Markov Decision Processes*. John Wiley & Sons, 1994.
9. David V. Pynadath, Milind Tambe, Yigal Arens, Hans Chalupsky, Yolanda Gil, Craig Knoblock, Haeyoung Lee, Kristina Lerman, Jean Oh, Surya Ramachandran, Paul S. Rosenbloom, and Thomas Russ. Electric Elves: Immersing an agent organization in a human organization. In *AAAI Fall Symposium on Socially Intelligent Agents: The Human in the Loop*, pages 150–154, 2000.
10. Paul Scerri, David V. Pynadath, and Milind Tambe. Adjustable autonomy in real-world multi-agent environments. In *Proceedings of the Conference on Autonomous Agents*, pages 300–307, 2001.
11. Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.
12. Daniel S. Weld and Oren Etzioni. The first law of robotics (a call to arms). In *Proceedings of the National Conference on Artificial Intelligence*, pages 1042–1047, 1994.

Formal Theories of Negotiation

Frank Dignum

Institute of Information and Computing Science, Utrecht University
3508 TB Utrecht, The Netherlands
dignum@cs.uu.nl

In any Multi-Agent System (MAS) the coordination between the agents is of crucial importance. The coordination can be programmed into the agents and the protocols that they use to communicate. However, to make full advantage of the fact that the agents are autonomous and pro-active we would like the interactions not to be completely fixed, but flexible and depending on the situation. In order to establish this the agent communication will have to be more peer-to-peer than hierarchical. I.e. agents can *request* other agents to perform some task or give some information but do not often have the power to *order* the other agent to perform the task.

In this situation the agents will have to "negotiate" in order to get to an agreement about the performance of the task. The negotiation serves to align the private goals of the two agents. In simple cases the goals are compatible and the agent that receives the request will just perform the task. In other cases the agents may have different goals and believes about the world and therefore the agent that receives the request has no incentive to perform the task at all. In this case the requesting agent should provide a justification for the other agent to perform the task.

The above illustrates that, although most people think about negotiation in the context of trade and making deals, negotiation also plays an important role in the "normal" coordination within a MAS. In general, negotiation can be defined as the process in which two or more parties attempt to reach a joint decision on issues in dispute. In order to accomplish this, the parties will strategically share information and search for alternatives. The nature of the actual negotiation of course depends on the level and way the information is shared and the type of agreements or joint decisions that are searched.

In the simple case of using Contract Net the negotiation only consists of one step in which several potential suppliers offer their services after a request for that service has been broadcast by the contractor. All the receivers of the request can decide whether they want to send an offer or not. I.e. whether the service would be in line with their current goal(s). If it is not they will not send an offer. If it is compatible but costs much trouble they will send an offer in which they request a high payment. Otherwise they can send an offer in which less payment is requested. The one who sends the best offer gets to perform the service against the requested payment.

This type of negotiation is often used as coordination mechanism, because it is easily implementable and costs relatively little overhead for the agents. The agents only have to compare the request with their own goals and reply appropriately. They will not (in general) change their own goals on the basis of the request for a service. This means that the strategy of the agent for the "negotiation" is also very simple.

Taking this example as a starting point we see that much work on formal negotiation is based on game theoretic research. The strategies are determined by the type of information that is known to both parties and the type of agreement that they try to establish. Some examples of this type of work in this volume are [3,5,6]. All of these papers take some situation and preconditions into account and try to find the best strategy for the agents in these conditions (usually based on a utility function). The advantage of the game theoretic approach is that the strategy can be modeled as a utility function that can be calculated by each agent. The utility function can be implemented in each agent (maybe with different weights) and afterwards the agents respond in a negotiation according to this function.

However, this approach is also limited to situations where agents do not change their utility functions during the negotiation or change their goals according to new information that they receive. For these situation one would like to have an argumentation based approach in which the goals of the parties are adjusted and the utility functions are changed according to arguments. Of course, this approach is more flexible than the game theoretical one, but also requires a heavier reasoning mechanism in the agents. One step towards this type of negotiation is taken in [4] in which dialogues for negotiation are discussed.

As can be seen from the number of references above, most work seems to take the game theoretical approach as their basis. However, it is very important to verify whether the presuppositions that are used in the game theoretic setting also hold in practice. In [1] some consequences are shown that arise when some preconditions do not hold in practice. In this paper, on the Vickrey auction, it is shown that if the utility of an agent depends on the utility of other agents it might bid higher than its private value for a product just to prevent a competitor to acquire the product. It is a perfect example of a precondition that does not hold in practice while being the cornerstone of the dominant strategy for the agents.

Another point to raise when discussing negotiation is who sets the rules of the negotiation. If one agent is trying to use the unconditional Contact Net protocol while the other agent uses the leveled commitment Contract Net protocol, they will have different perceptions of the situation after the first exchanges. In many situations it is necessary to have a third party that functions as an intermediary and which determines which rules are used and which protocol is followed during the negotiation. In many MAS this third party is the designer of the whole system who determines the negotiation protocols that are used in the system. However, this will only work in closed systems where all agents are designed by the same party. In open systems (on the Internet) it is no longer obvious which negotiation protocols can be used. In these situations electronic institutions take the role of trusted third party that determines the interaction patterns according to which the parties negotiate and close deals. In order for agents to be able to function in the institution they have to "know" the interaction protocols that are used in that institution. Therefore the institution has to be described formally such that an agent can read this description and decide how it should act within that institution. This is one of the aims of the work described in [2].

Some topics that are not explicitly addressed in the papers in this volume are e.g. the role of the communication language. In [4] a logical communication language is used, but little attention is given to the types of performatives that are available. Should there be special performatives available in the language connected to the negotiation protocols (such as "bid" for an auction)? Allowing this type of performatives in a language like the

FIPA ACL would make them much more efficient in negotiation situations, but might also lead to many ad hoc message types without a clear semantics.
Related to this issue is that of ontologies. How do the agents know what the other agent is offering? When agents negotiate about a set of known tasks or products this is not an issue, but in open systems products and tasks might be offered in a way not known to all agents.
Usually the above topics are not considered in the negotiation theories, because these theories somehow suppose that all parties have a common protocol and common knowledge about the tasks and/or products about which they negotiate. The theories mainly deal with the strategies that agents use to generate the next offer in the negotiation. However, in many "real" negotiations these issues are crucial for the final result and cannot be left undiscussed!
Hopefully the papers in the special negotiation session in this volume indicate some steps forward in the research in this field and lead to research about negotiation in more and more open (read: practical) situations.

References

1. F. Brandt and G. Weiss. Antisocial agents and Vickrey Auctions, In *this volume*.
2. M. Esteva, J. Padget and C. Sierra. Formalizing a Language for Institutions and Norms, In *this volume*.
3. P. Faratin, M. Klein, H. Sayama and Y. Bar-Yam. Simple, Negotiation Agents in Complex Games, In *this volume*.
4. F. Sadri, F. Toni and P. Torroni. Dialogues for Negotiation: Agent Varieties and Dialogue Sequences, In *this volume*.
5. S. Shaheen, M. Wooldridge and N. Jennings. Optimal Negotiation Strategies for Agents with Incomplete Information, In *this volume*.
6. P. Stone and M. Littman Implicit Negotiation in Repeated games, In *this volume*.

A Stable and Feasible Payoff Division for Coalition Formation in a Class of Task Oriented Domains[1]

Maria Victoria Belmonte, Ricardo Conejo, José Luis Pérez-de-la-Cruz, and Francisco Triguero

Dpto. Lenguajes y Ciencias de la Computación, Universidad de Málaga
Campus de Teatinos, P.O.B. 4114, 29080, Málaga, Spain
{mavi,conejo,perez}@lcc.uma.es

Abstract. In the last few years the use of coalition formation algorithms in multi-agent systems has been proposed as a possible way of modelling autonomous agent cooperation. Game theory provides different concepts for the stability of solutions in cooperative games, regarding the fairness of the resultant payment configuration. One of these is the core. In this paper we present an analysis -based upon game theory- for a class of task-oriented problems arising from some Internet transactions. A procedure that guarantees an optimum task allocation is defined, and a new payoff division model is proposed for the corresponding value. We prove that the proposed payoff division lies inside the core. The whole computation (of the optimal value and the payoffs) has a polynomial complexity in the number of agents.

1 Introduction

Automatic negotiation mechanisms among self-interested agents have become an important research topic in the multi-agent system (MAS) community. The huge development of communication platforms and technologies [1;2;3] in which independent agents can interact with each other carrying out secure electronic commerce transactions, has given rise to a great demand on negotiation computational support. In all these cases, multi-agent technology tries to automate the coordination activities among autonomous agents, by means of a negotiation software attached to each of them. This negotiation software can provide great savings in terms of time and costs, and can discover better solutions than human negotiators can in strategically complex environments. Auctions, bargaining, markets and coalitions [4] have been some of the most used technologies.

More concretely, in the case of the coalition formation problem, many different approaches have been proposed in the MAS community. The objective of the coalition formation process is to find -within a reasonable time and spending a reasonable amount of computational efforts- a stable payment configuration which maximizes the agents' payoff, improving their benefits and abilities to satisfy their goals.

[1] This work has been partially supported by the Spanish Government under grant CICYT TIC2000-1370-C04-02

Game theory provides different concepts (core, kernel, Shapley value,…) [5;6] for defining the stability and fairness of the payment distribution among players in N-personal games. In these models, each of the participants tries to maximize its own utility, knowing that all the agents will adopt the same attitude. These models have been used in the MAS field, often for *superadditive* environments [7;8;9]. This term is used in game theory to name those environments where the global coalition is always the most profitable; in this case, we say -informally speaking- that cooperation costs do not grow significantly with the number of cooperating agents i.e., that communication and computational costs are small. Other MAS solutions for general environments (super and non-superadditive) are presented in [10;11]. In the first one, Sandholm et al. use the computing costs as the term that modifies the utility of the rational agent, extending in this way the game theory framework, and presenting a coalition formation model for limited agents. However, at the meta-level i.e., in the determination of the computational costs, unbounded rationality is used (without costs and instantaneously). In the second one [11], Shehory et al. present an anytime coalition formation algorithm. In this algorithm a modified game theory solution concept, the *kernel*, is used in order to define the stability of the resulting utility distribution obtaining, at the same time, a polynomial complexity. However, in order to achieve this, the number of agents in each coalition is limited, affecting the optimality of the result. Later, it was shown in [12] that any general algorithm for coalition formation, guaranteeing a solution inside a neighbourhood of the best one, in terms of utility maximization, has to search in an exponential number of possible solutions.

On the other hand, coalition formation for electronic markets has also been considered in the literature [13;14]. Coalition formation is seen as a mean to aggregate groups of customers coming together to procure goods at a volume discount and incentives for the creation of such groups. The first approach [13], provides and criticizes several coalition models to form *buying clubs*, but does not concentrate on their formal analysis, in the sense of coalition stability or optimality of the solution. The second one [14], proposes a physics-motivated mechanism for coalition formation, where the agents may leave coalitions with some probability. The model is formally expressed as a series of differential equations and has a low complexity, but -as far as we know- a detailed stability analysis of the system has not been carried out yet.

In this paper we analyse and model a kind of electronic transactions in Internet. We assume that there is an indefinite number of agents that can communicate with negligible cost. Each of them must perform initially a certain amount of a task; but -since each of them performs the task at a different unitary cost- they can interchange tasks in order to achieve a better global efficiency. Our solution guarantees -given the assumptions stated in 2.2- an optimum task allocation among coalition members and a stable payoff division (according to the core). In addition, the computation of the solution has a polynomial complexity, so it is computationally feasible.

The article is organized as follows: in the next section we show the formalism and assumptions used to model the work domain (a class of task oriented domains). In the third section, we solve the optimization problem i.e., the problem of distributing the global task among coalition members, in such a way that the coalition utility is maximized. In section four, we propose the payoff division model and prove its stability according to the core. We also prove the stability of the payoff division for two particular classes of problems, which can be included into the general one. We will finish, in the fifth section, with some conclusions and future works.

2 Framework Description

In this section we give the formal definition of the problem at hand, taking into account that we are modelling coalition formation for a class of task oriented domains. Then, we also describe some of the game theory definitions and concepts that we use in the model, and present some modifications, in order to consider the specific properties of our problem.

2.1 Task Oriented Domains

As defined in [15], in a *Task-Oriented Domain* (*TOD*) an agent's activity is described in terms of a set of tasks that have to be achieved by itself. A *task* is considered as an infinitely divisible job that has to be carried out. Each agent has the necessary capabilities or resources to carry out the task, so it can be accomplished without the assistance of other agents. However, it is possible to form a coalition among agents with a new distribution of the task that may allow them to obtain benefits. So, in task oriented domains, agent's activity is modelled in a simple way, but it can be useful in order to describe problems which naturally fit to this kind of domains, or to analyse simplified versions of more complicated ones.

In addition it is often assumed that the following features characterize the situation:

- Agents are rational or self-interested, so they try to maximize their own utility.
- The tasks can be transferred among agents, and they use a monetary system to evaluate them. This transference needs time and effort by the agents (cost). In addition, the payment reachable by any coalition can be distributed among the agents that form it.
- The information about tasks, capabilities and costs of each agent is accessible to the remaining ones.
- Superadditive environments. Usually it is argued that almost all games are superadditive because, at worst, the agents in the global coalition can use the solution that they had reached in separate coalitions. But some games may result non superadditive due to the cost of the coalition formation process itself (coordination and communication costs). In our case, we will suppose superadditive environments, as justified in the following paragraph.

2.2 Assumptions and Definitions

Let us consider a set of *n* agents $N=\{a_1,a_2,..,a_n\}$ that can communicate via Internet and such that each agent a_i must perform a certain initial amount t_i of task units. The agent a_i can perform a maximum of k_i task units ($t_i \leq k_i$) and the unitary cost is c_i. We will define the surplus capacity h_i of a_i by $h_i=k_i-t_i$ $(0\leq h_i)$. The agents can communicate and change the amounts of tasks initially assigned to each one, in order to achieve a better global efficiency; let $t_{i,j}$ be the number of transferred task units from a_i to a_j. However, to allow this new deal, they must incur in a transfer cost; namely, we will assume that the cost of transferring a task unit from a_i to a_j is $c_{i,j}$. In this way, the unitary profit, $b_{i,j}$, attached to the transfer of a task unit between a_i and a_j is $b_{i,j}=(c_i-c_j)-c_{i,j}$ and $B_{i,j} = b_{i,j} * t_{i,j}$ is the transfer profit. We will assume that transfer costs satisfy the triangular inequality i.e., $c_{i,j} + c_{j,k} \geq c_{i,k}$ $\forall i,j,k \in N$. On the other hand, we will assume that coalition formation costs (communication and coordination) are negligible compared to actual task transfer costs (physical transfer of materials). This assumption could sometimes be justified by the decreasing costs (economic, opportunity, etc.) of asynchronous Internet communications and leads to a superadditive problem i.e., a problem in which the incorporation of a new agent to a coalition does not imply any cost.

Coalition formation is often studied [5;6] using the concept of *characteristic function games*. In such games, the value of each coalition *S* is calculated by a characteristic function *v(S)*, in such a way that the coalition value is independent of the non-members actions. Definitions 1 and 2 are standard in the literature:

Definition 1. *A coalition S is a subset of the set of players N, that is able to reach a binding agreement.*

Definition 2. *The characteristic function with transferable utility or coalitional value, v(S), of a game with a set of players N, is a scalar function that associates* $v(S) \in \Re$ *to each* $S \subset N$.

So, a coalition *S* is a group of agents that decide to cooperate and *v(S)* is the total utility that the members of *S* can reach by coordinating themselves and acting together. Henceforth, we will use the notation *(N,v)* to denote a game in which *N* is the set of agents and *v* is the characteristic function or coalitional value.

We introduce the terminology of definitions 3 and 4 in order to perform the theoretical analysis of our model:

Definition 3. *(Representing function). Let (N,v) be a superadditive game and* $N=\{a_1,a_2,..,a_n\}$ *be the set of agents with* $n=|N|$. *Let us assume that each agent is described by a finite set of m attributes* $W=\{w_1,w_2,..,w_m\}$ *whose values for an agent* a_i *are* $\{w_{i1},w_{i2},..,w_{im}\}$. *We say that the function* $V(w_{11},w_{12},..,w_{1m}, ..., w_{n1},..,w_{nm})$ *represents the characteristic function v(S) iff* $\forall S$ $v(S) = V(w'_{11},w'_{12},..,w'_{1m}, ..., w'_{n1},..,w'_{nm})$, *where* $w'_{ij}= w_{ij}$ *when* $a_i \in S$ *, 0 otherwise.*

In our model, we will choose the initial amount of tasks and the initial surplus capacity as the two attributes that describe an agent i.e., in the terminology of definition 3, $m=2$, $w_{i1}=t_i$, $w_{i2}=h_i$.

Definition 4. *(H1C condition). Let V be a representing function. We say that V satisfies the H1C condition iff (i) V is homogeneous of degree 1 i.e.,* $\forall \mathbf{w} \in \Re^{+nm}$ $\forall \lambda \in \Re^{+}$ $V(\lambda \mathbf{w}) = \lambda V(\mathbf{w})$; *(ii) V is continue; (iii) V is concave, i.e.,* $\forall \mathbf{w}_1, \mathbf{w}_2 \in \Re^{+nm}$ $\forall \lambda, 0\leq\lambda\leq 1,$ $(1-\lambda)V(\mathbf{w}_1) + \lambda V(\mathbf{w}_2) \leq V((1-\lambda)\mathbf{w}_1 + \lambda \mathbf{w}_2)$.

In our domain the coalitional value of any coalition will be stated as the addition of all the transference profits, $B_{i,j}$, associated to each task transference $t^*_{i,j}$ $\forall i,j \in S/i \neq j$, obtained from an optimum task allocation T^*:

$$v(S) = \sum_{i,j \in S/i \neq j} B_{i,j} = \sum_{i,j \in S/i \neq j} b_{i,j} * t^*_{i,j} \text{ , } \forall i, j \in S / i \neq j \tag{1}$$

where $T^*=(t_1^*,t_2^*,\ldots,t_n^*)$ is the optimum task allocation vector, $t_i^* = t_i + t^*_{i,j}$, i.e. t_i^* is the number of tasks that a_i has to carry out after the redistribution of the tasks between the set of agents inside S in order to achieve a maximum of efficiency.

Alternatively, the coalitional value can be calculated as the difference between the initial costs that the coalition members must pay to carry out their tasks, minus the cost of the optimum task allocation obtained after the redistribution of these. This cost is formed by the addition of two different costs: the task execution cost c_i, and the transfer cost $c_{i,j}$.

$$v(S) = \sum_{i \in S} t_i c_i - \sum_{i,j \in S / i \neq j} (t_i^* c_i + c_{i,j} t^*_{i,j}) \tag{2}$$

3 Solving the Task Allocation Problem

Solving the optimum task allocation or distribution among the coalition members, with the goal of maximizing coalition profits, is a non-trivial problem. Moreover, in many domains this problem may result too complex from the combinatorial point of view. However, in our case, an algorithm that solves the optimization problem of each coalition and is computationally feasible can be implemented.

The goal of our algorithm will be to establish the optimum task allocation vector, T^*, i.e. the allocation that minimizes the costs (the costs to carry out the tasks plus the transfer costs). In order to establish T^* we have to calculate the number of transferred task units $t^*_{i,j}$, which yields this optimum allocation. For this reason, we state the task allocation as a linear programming problem, in which the costs are minimized subject to the capacity constraints of each agent. The number of possible transferences, $t_{i,j}$, will be n^2 (further simplifications can be made in order to reduce this number to $n(n-1)/2$).

The linear programming problem statement will be the following one (notice that there are n^2 variables and $3n$ constraints):

$$MIN\left(\sum_{i \in N} c_i * t_i + \sum_{i,j \in N / i \neq j} c_{i,j} * t_{i,j}\right) \tag{3}$$

$$\begin{aligned} &\textit{subject to} \\ &\quad t_i \leq k_i, \forall i \in N \\ &\quad 0 \leq t_i, \forall i \in N \end{aligned}$$

Where $t_i \in N$ is the task that a_i has to carry out after the task redistribution among the agents. It is calculated as the initially assigned task to a_i, t^o_i, minus the task transferences going out of a_i, plus the task transferences going into a_i:

$$t_i = t_i^o - \sum_{j \in N \,/\, j \neq i} t_{i,j} + \sum_{j \in N \,/\, j \neq i} t_{j,i} \ , \ \forall i \in N \tag{4}$$

In this statement the only unknown quantities are the transferred tasks among agents, $t_{i,j}$. So, when we solve the problem and the transfer values are known, it will be possible to calculate the optimum task allocation $t_i^* \ \forall i \in N$, and hence solve the optimization problem of the grand coalition. Well-known algorithms [16] (Karmarkar's one) solve this problem in polynomial time (worst-case) in terms of the number of variables and constraints, hence in the number of agents, *n*.

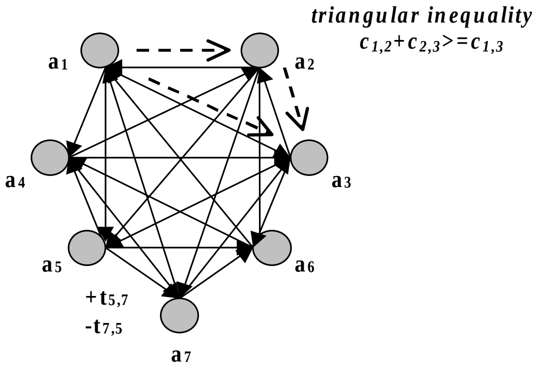


Fig. 1. Graph that represents all the possible task transferences among the agents

Graphically (figure 1), we can show the model like a graph in which each agent is connected to the *n-1* remaining ones. As a result of the task allocation the agents are classified in two types: *producers* and *consumers*. Producer agents are agents that only transfer tasks and consumer agents are agents that only receive tasks.

If we impose the following constraint: $c_{i,j} + c_{j,k} \geq c_{i,k} \ \forall i,j,k \in N$ called *triangular inequality*[2], there is a solution in which any agent may be a producer or a consumer but not both. In fact, let us suppose two transfers, the one of t_1 units from a_i to a_j, the other of t_2 units from a_j to a_k. Let us call $t_3 = min(t_1, t_2)$ and assume $t_3 = t_1$. Then, the transfer of t_3 units from a_i to a_k and the transfer of $t_2 - t_3$ units from a_i to a_j is an assignment at least as good as the original one.

4 Payoff Division According to the Core

Payoff division is the process of dividing the utility or benefits of a coalition among its members. More formally, let us state the following standard definitions:

[2] Even if this constraint is not fulfilled, it would be possible to apply this algorithm by means of the inclusion of virtual costs [19]

Definition 5. *A coalition structure CS = (T_1,....,T_m) is a partition of the set of players into coalitions. Each set T_k is not empty, $T_i \cap T_j = \varnothing$ for all i, j ∈ 1,....,m, with j ≠ i and $\cup_{k \in N} T_k = N$. This is a non-empty and mutually disjoint partition of the set of agents.*

Example: in a game with three agents, there are 7 possible coalitions: {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}, and 5 coalition structures: {{1},{2},{3}}, {{1,2},{3}}, {{1,3},{2}}, {{2,3},{1}}, {{1,2,3}}.

Definition 6. *A Payment Configuration, PC, is a pair (**x**,CS) where **x**=(x_1,x_2,..,x_n)$\in \Re^n$ is a payment vector and CS is a coalition structure. A PC is not more than a concrete payment vector **x**, paired with a concrete coalition structure CS. Every coalition has to receive a payment that must be equal to what the coalition will be able to get.*

The problem is to distribute the utility of the coalition in a fair and stable way, so the agents in the coalition are not motivated to abandon it. Game theory provides different concepts (core, kernel, Shapley value,...) for the stability of coalitions in cooperative games, regarding the fairness of the resultant payment configuration. However, from the point of view of multi-agent systems, these solutions raise two kinds of problems: 1) Usually it is too complex to check the definition for a given payoff division; 2) They abstract from any underlying bargaining process, so the theory fails to explain how the players reach any solution or equilibrium. Exceptions are dynamic theories or transfer schemes for coalition formation, for example Stearn’s transfer scheme for the kernel [17], Wu’s transfer scheme for the core [18],… However, these schemes may require an infinite number of iterations in order to converge, and the computational complexity may be exponential again.

Some prior works have solved these problems in particular cases. For example, Rosenschein and Zlotkin in [7] give a mechanism for payoff division in concave superadditive TODs, and they prove that the calculated payments are the Shapley values for each agent. However, their result is only probabilistic i.e., it only holds for expected values, not for real ones.

On the other hand, the core is the strongest of these classical solution concepts and in the general case may include none, one or many payoff vectors. Formally:

Definition 7. *The core of a characteristic function game (N,v) is the set of all the PCs (**x**,CS), if any, so that no subgroup of agents is motivated to abandon the coalition structure CS, so:*

$$core = \{(x, CS) \mid \forall S \subseteq N, v(S) \le \sum_{i \in S} x_i \text{ and } v(N) = \sum_{i \in N} x_i\} \tag{5}$$

In order to guarantee the stability of the payoff division, we propose for our problem a new method, which computes a stable payoff division in the sense of the core. The method is based upon lemma 1:

Lemma 1. *If V represents v(S) and satisfies the H1C condition, then the payment vector given by*

$$x_i = \sum_{j=1}^{m} w_{ij} \frac{\partial V}{\partial w_{ij}}, \ \forall i \in N \tag{6}$$

(where all the derivatives are computed in the same region R_i) lies inside the core.

Proof. The proof is straightforward. In fact, homogeneity of degree 1 ensures global rationality i.e., along the line from **0** to ***x*** there exists the directional derivative that could be computed from the partial derivatives inside a certain region and must be 1. That proves that: $\sum_{i \in N} x_i = v(N)$.

On the other hand, coalitional rationality can be derived from the concavity of *V*. Let w^0_{ij} be the vector of attributes for all the agents in *N*, and for each subcoalition $S \subset N$ let $w'_{ij} = w^0_{ij}$ when $a_i \in S$, $w'_{ij} = 0$ otherwise. Then:

$$A = \sum_{i \notin S} x_i = \sum_{i \notin S} \sum_{j=1}^{m} w^0_{ij} \frac{\partial V}{\partial w_{ij}} \leq V(\mathbf{w}^0) - V(\mathbf{w}') \text{ (by concavity of } V) \tag{7}$$

But, since *V* represents *v*, $V(\mathbf{w}^0) = v(N)$ and $V(\mathbf{w}') = v(S)$, hence $A \leq v(N) - v(S)$. But then $\sum_{i \in S} x_i = v(N) - \sum_{i \notin S} x_i = v(N) - A \geq v(S)$, q.e.d.

Now, in order to apply lemma 1 to our problem, we must prove the following proposition:

Proposition 1. *Let us consider a problem of the class defined in section 2. There exists a function $V(t_1, h_1, ..., t_n, h_n)$ that (i) V is computable in a polynomial time in n; (ii) V represents v(S); and (iii) V satisfies the H1C condition.*

Sketched proof. First, the linear model (3)-(4) presented in section 3 ensures the existence and feasibility of *V* as a function of the parameters $t_1, h_1, ..., t_n, h_n$. Obviously, if some of the parameters are set to 0, the linear model represents a subcoalition $S \subset N$ and computes the optimum value *v(S)*. In virtue of physical considerations, it is easy to accept that *V* is homogeneous of degree 1, i.e., scaling all the parameters by a factor of α yields a new value of *V* also scaled by α. Analogously, physical considerations show that the partial derivatives are always non increasing and *V* is concave.

More formally, Mills' theorems [20] ensure that *V* can be expressed by a family of linear combinations of $t_1,h_1, ..., t_n,h_n$; and there exists a linear finite partition $\{R_1,R_2,..,R_k\}$ of $\mathfrak{R}^{+nm}$ such that *V* is linear in each R_i, $1 \leq i \leq k$.

Proposition 2. *Let $\mathbf{w}_0$ be the vector of initial tasks and surpluses. Let R_0 be a region of the partition where $\mathbf{w}_0$ lies. Then the payment vector given by*

$$x_i = t_i \frac{\partial V}{\partial t_i} + h_i \frac{\partial V}{\partial h_i}, \ \forall i \in N \tag{8}$$

(where all derivatives are computed in R_0) lies inside the core and is computable in a polynomial time in terms of the number of agents, n.

Sketched proof. By lemma 1 and proposition 1, ***x*** lies inside the core. By Mills' theorem, given the set of solutions to (3)-(4) and to its dual program, there is an effective procedure to compute the partial derivatives.

Proposition 2 justifies the claim in the title of this paper. Notice, that the linear program (3)-(4) always has a solution. In fact, the convex region defined by (3)-(4) is always bounded and not void.

On the other hand, the derivatives given in (8) are always well defined. Hence, we can prove that -in this particular TODs- the core is not empty, since we can always calculate a payment vector that belongs to the core.

Let us now consider a couple of simplified cases with an easy intuitive interpretation. First let us suppose null transfer costs ($c_{i,j}=0, \forall i,j \in N$) and let us assume that the agents (all with different costs) are ordered by cost ($c_1>c_2>\ldots>c_n$). That ordering can always be achieved in polynomial time. In this case, since there are not transfer costs, the optimum task allocation will consist of transferring all the task units to the agent with the lower cost. But, as each agent has a limited capacity, in most of the situations all the agents are not be able to transfer all their tasks to a_n (agent with the lower cost). Then, and in order to maximize agents' utility, the agent with the higher cost will transfer to a_n, the agent with the lower cost, all the task units that this one can receive. Afterward, the second agent with the higher cost will try to transfer its entire task to the agent, which has, in that moment, the lower cost and a surplus of capacity. The process continues in this way, and finally we will have a sub-group of producer agents, which transfer all their tasks, a sub-group of consumer agents, which receive tasks and consume all their surpluses, and a unique fringe agent, a_M, which also receives tasks but it does not consume all its surplus. The coalition value of the global coalition will be stated as the addition of all the global profits $B_{i,j}$ attached to each task transference $t_{i,j}$ from a_i (producer) to a_j (consumer) $\forall\, i,j \in N / i \neq j$:

$$v(N) = \sum_{i,j \in N / i \neq j} B_{i,j} = \sum_{i,j \in N / i \neq j} b_{i,j} * t_{i,j} \qquad \textbf{(9)}$$

By means of elementary manipulations, we can establish *v(N)* as:

$$v(N) = \sum_{i=1/i<M}^{n} t_i (c_i - c_M) + \sum_{i=1/i>M}^{n} h_i (c_M - c_i) \qquad \textbf{(10)}$$

And applying the results of lemma 1, the payment vector ***x*** will be given by:

$$\begin{aligned} x_i &= t_i (c_i - c_M)\, , \text{ if } a_i \in \text{producers} \\ x_i &= h_i (c_M - c_i)\, , \text{ if } a_i \in \text{consumers} \\ x_i &= 0\, , \text{ if } a_i = a_M \end{aligned} \qquad \textbf{(11)}$$

Additionally, let us now suppose a non-limited capability vector i.e.,$\forall i \in N,\ k_i = \infty$. In this case, since each agent has an infinite capability and there are not transfer costs, the optimum task allocation will consist of transferring all the task units to the agent with the lower cost. Thus, as we consider the agents ordered by cost, all the agents, except a_n, will transfer their task to a_n. The coalition value of the global coalition will be stated as the addition of all the global profits $B_{i,n}$ attached to the transference of all the tasks $t_{i,n}$ $\forall i\in N/i\neq n$ from a_i to a_n:

$$v(N) = \sum_{i\in N/i\neq n} B_{i,n} = \sum_{i\in N/i\neq n} t_{i,n} * b_{i,n} = t_{i,n}(c_i - c_n) \quad \textbf{(12)}$$

And applying the results of lemma 1, the payment vector ***x*** will be given by:

$$x_i = t_i(c_i - c_n),\ \forall i\in N \quad \textbf{(13)}$$

5 Conclusions and Future Works

Coalition formation is an important method for cooperation in multi-agent systems. Autonomous agents forming coalitions may improve their benefits and abilities to satisfy their goals. The objective of a coalition formation process is to find a stable payment configuration, which maximizes the agents' payoff and doing it within a reasonable time and computational efforts.

In this paper we have shown a new payoff division model according to the core, for coalition formation among self-interested agents in certain superadditive task oriented domains. The model has a polynomial complexity and guarantees an optimum task allocation. The optimization within the coalition is stated as a linear programming problem, in which the costs are minimized subject to the capacity constraints of each agent. Furthermore, we have proved the stability of the proposed payoff division in the sense given by the core. This kind of stability has also been demonstrated for the payoff division of two particular types of problems, which may be included into the general one. These payment vectors are computable in polynomial time in the number of agents, *n*.

We are currently developing a software architecture to implement and check the coalition formation model in real environments. We have chosen the FIPA-OS agent platform [1] to construct an open multi-agent system, in which a new layer is added to the FIPA-OS agents providing a new service, coalition formation.

Future research will include the extension of our model to other more realistic cases. We are planning to impose new constraints, as limiting the number of transferred tasks among agents, or considering the existence of different kinds of tasks grouped under the same production capacity, in such a way that a broader range of electronic commerce problems could be modelled.

References

1. Poslad, S., Buckle, P., and Hadingham, R. *The FIPA-OS agent platform: Open Source for Open Standards*. In proceedings of Autonomous Agents AGENTS-2000, Barcelona, 2000.
2. Nwana H. S., Ndumu, D.T, Llee, L.C., and Collis, J. C. *ZEUS: a Toolkit for Building Distributed Multi-Agent Systems*. Applied Artificial Intelligence Journal Vol. 13(1), 129-186, 1999.

3. Bellifemine, F., Poggi, A. and Rimassa, G. *JADE-A FIPA-compliant agent framework*. In Proceedings of PAAM'99, London, 97-108, 1999.
4. Weiss, G. (eds.). Multi-agent Systems. *A modern Approach to Distributed Artificial Intelligence*. Sandholm, T. *Distributed Rational Decision Making*. MIT Press, 1999.
5. Kahan, J., and Rapaport, A. *Theories of Coalition Formation*. Lawrence Erlbaum Associates Publishers, 1984.
6. Aumann, R.J., and Hart, S. (eds.) *Handbook of Game Theory with Economics Applications*. North-Holland, 1992.
7. Rosenschein, J.S., and Zlotkin, G. *Coalition, Cryptography, and Stability: Mechanisms for Coalition Formation in Task Oriented Domains*. In proceedings AAAI-94, Seattle, WA, 432-437, 1994.
8. Ketchpel, S. *Forming Coalitions in the face of uncertain rewards*. In proceedings AAAI-94, Seattle, WA, 414-419, 1994.
9. Klush, M., and Shehory, O. *Coalition Formation among rational information agents*. Lecture Notes in Artificial Intelligence, No. 1038, Agents Breaking Away. Van de Velde, W., and Perram, J.W. (eds.), 204-217, Springer-Verlag, 1996.
10. Sandholm, T., and Lesser, V. *Coalitions among computationally bounded agents*. Artificial Intelligence 94, 99-137, 1997.
11. Shehory, O., and Kraus, S. *Feasible Formation of coalitions among autonomous agents in non-super-additive domains*. Computational Intelligence Vol. 15(3), 218-251, August, 1999.
12. Sandholm, T., Larson, K., Anderson, M., Shehory, O., and Thome, F. *Coalition Structure Generation with Worst Case Guarantees*. Artificial Intelligence 111(1-2), 209-238, 1999.
13. Tsvetovat, M., Sycara, K., Chen, Y., and Ying J. *Customer Coalitions in Electronic Marketplace*. In proceedings of AGENTS-2000, Barcelona, 263-264, 2000.
14. Lerman, K. and Shehory, O. *Coalition Formation for Large-Scale Electronic Markets*. In proceedings of ICMAS-2000, Boston MA, 2000.
15. Rosenschein, J.S., and Zlotkin, G. *Rules of Encounter*. MIT Press, 1994.
16. Schrijver, A. *Theory of Lineal and Integer Programming*. John Wiley & Sons, N.Y., 1986.
17. Stearns, R.E. *Convergent transfer schemes for n-person games*. Transactions of the American Mathematical Society 134, 229-459, 1968.
18. Wu, L.S. *A dynamic theory for the class of games with nonempty cores*. SIAM Journal of Applied Mathematics, 32, 328-338, 1977.
19. Belmonte, M.V., Conejo, R., Perez-de-la-Cruz, J.L., and Triguero, F. *A Coalition Formation Model for a Class of Task Oriented Domains*. Technical Report, n. LCC-ITI-2001/7, 2001
20. Mills, H. D. *Marginal values of matrix games and linear programs*. In Kuhn, H. W. and Tucker, A. W. (eds.): Linear inequalities and related systems. Princeton University Press, Princeton, 1956.

Antisocial Agents and Vickrey Auctions

Felix Brandt and Gerhard Weiß

Institut für Informatik, Technische Universität München
80290 München, Germany
Tel. +49-89-28928416 / 28922606
{brandtf,weissg}@in.tum.de

Abstract. In recent years auctions have become more and more important in the field of multiagent systems as useful mechanisms for resource allocation and task assignment. In many cases the Vickrey (second-price sealed-bid) auction is used as a protocol that prescribes how the individual agents have to interact in order to come to an agreement. We show that the Vickrey auction, despite its theoretical benefits, is inappropriate if "antisocial" agents participate in the auction process. More specifically, an antisocial attitude for economic agents that makes *reducing the profit of competitors* their main goal besides maximizing their own profit is introduced. Under this novel condition, agents need to deviate from the dominant truth-telling strategy. This paper presents a strategy for bidders in repeated Vickrey auctions who are intending to inflict losses to fellow agents in order to be more successful, not in absolute measures, but relatively to the group of bidders. The strategy is evaluated in a simple task allocation scenario.

1 Introduction

The area of multiagent systems [20], which is concerned with systems composed of technical entities called agents that interact and in some sense can be said to be intelligent and autonomous, has achieved steadily growing interest in the past decade. Two key problems to be addressed in this area are automated resource allocation and task assignment among the individual agents. As a solution to these problems it has become common practice to apply well known results and insights from auction theory (e.g., [9,10]) and well understood auction protocols like the English auction, the Dutch auction, and the Vickrey auction. Among the different protocols, the Vickrey auction [18] (also known as second-price sealed-bid auction) has received particular attention within the multiagent community and has been applied in a variety of contexts like e-commerce, operating systems, and computer networks (e.g., [17,6,5,19,8,4]). The Vickrey auction is favored because of three main characteristics:

- it requires low bandwidth and time consumption
- it possesses a dominant strategy, namely, to bid one's true valuation (see Section 2)
- it is a sealed-bid auction (bids (expressing private values) remain secret)

These characteristics make the Vickrey auction protocol particularly appealing from the point of view of automation. The Vickrey auction, in its original formulation and as it is used for *selling goods* or *resource allocation*, works as follows: each bidder makes a sealed bid expressing the amount he is willing to pay, and the bidder who submits the highest bid wins the auction; the price to be payed by the winner is equal to the second highest bid. In *task assignment scenarios* the Vickrey auction works exactly the other way round (and for that reason is often referred to as reverse Vickrey auction): each bidder willing to execute a task makes a bid expressing the amount he wants to be payed for task execution, and the bidder submitting the lowest bid wins the auction; the winner receives an amount equaling the second lowest bid (and his payoff thus is the second lowest bid minus his prime costs for execution). If there is more than one winning bid, the winner is picked randomly from the highest bidders. This paper concentrates on the use of the Vickrey auction for task assignment scenarios; it should be noted, however, that all presented considerations and results do also hold for Vickrey auctioning in its original formulation. It is an important fact, that bids are sealed. Bidders that did not win an auction do not necessarily get to know who placed the lowest bid and particularly how low this bid was. The only information that is revealed to all bidders is the selling price (see [2] for a more detailed discussion of privacy and security issues in Vickrey auctions).

A general assumption often made in applications of multiagent systems is that an individual agent intends to maximize his profit without caring for the profits of others. This, however, is by no means the case in real-world settings. What can be often observed here is that a company, besides maximizing its own profit, deliberatively inflicts losses to rivaling companies by minimizing their profits. In fact, it is real-world practice that a company accepts a lower profit or is even willing to sell goods at a loss if this financially damages a competing company or at least helps to bind available and gain new costumers. Such a company obviously exhibits an *antisocial* behavior and attitude in the sense that it considers (at least for some period of time or in some cases) its absolute profit not as important as its profit relative to other companies' profits. (It is worth to point out that an "antisocial company" behaves rationally given its objective, even if it could be said to act irrationally under the condition that its objective were to maximize its absolute profit.) This paper investigates the performance of Vickrey auctioning in multiagent systems in which the presence of antisocial agents can not be excluded.

A related problem that has been discussed in the economic literature are "externalities" between bidders [7]. Externalities describe preferences of a bidder specifying who he wants to win an auction (besides himself), e.g., when selling patents or nuclear weapons. However, our approach is solely profit-oriented. Antisocial agents do not care who buys a good or who is awarded a task contract. They are just interested in decreasing their fellows' profits to strengthen their own market position. This is a major difference.

The paper is structured as follows. Section 2 explains why agents aiming at a maximization of their absolute profits should bid their true valuations in Vickrey auctions. Section 3 formally captures the antisocial attitude sketched above and Section 4 shows its impact on the bidding strategy. Section 5 introduces and analyzes an antisocial strategy for repeated Vickrey auctions. Section 6 presents experimental results that illustrate the implications of this strategy. Finally, Section 7 concludes the paper with a brief overview of advantages and disadvantages of the agents' "bad attitude".

2 The Dominant Strategy

The Vickrey auction has a dominant strategy, which means that if an agent applies this strategy he receives the highest possible payoff, no matter which strategies are used by the other bidders. The dominant strategy is to bid one's true valuation of the task. Even if an agent knows all the other bids in advance, he still does best by bidding his private valuation. The conditions for the existence of the dominant strategy equilibrium are that the bidders are symmetric (i.e., they can not be distinguished) and have independent valuations of the tasks[1]. This implies that tasks cannot be recontracted.

Given two agents A and B, their corresponding private values v_a and v_b, and their submitted bids b_a and b_b, the profit for agent A is defined by the following equation.

$$\mathit{profit}_a(b_a, b_b) = \begin{cases} b_b - v_a & \text{if } b_a \leq b_b \\ 0 & \text{if } b_a \geq b_b \end{cases} \tag{1}$$

It can easily be seen why bidding the task value is the optimal strategy. We consider agent A and investigate the possible profits he would make by not bidding his private value. It suffices to model only one opposing agent B representing the entire competition because A only cares whether he wins or loses. He does not draw distinctions between his fellow bidders.

If A bids more than his prime costs ($b_a > v_a$) there are three subcases conditional on agent B's bid b_b:

i) $b_b < v_a < b_a$: B wins the auction and receives more money than if A had bid v_a.
ii) $v_a < b_b < b_a$: A loses the auction, instead of winning it by bidding v_a.
iii) $v_a < b_a < b_b$: A still wins the auction, but does not gain anything, because the task price remains b_b and his payoff is still $b_b - v_a$.

If he bids $b_a < v_a$ the following cases describe the resulting situations:

iv) $b_a < v_a < b_b$: A wins, but is paid the same amount of money (b_b) as if he bid v_a.
v) $b_a < b_b < v_a$: A wins, but gets less money than his own prime costs, i.e. he is losing $v_a - b_b$.
vi) $b_b < b_a < v_a$: A still loses and reduces B's payoff by $v_a - b_a$.

Concluding, bidding anything else than v_a cannot yield more profit than bidding the true valuation v_a. Obviously, this extremely simplifies the bid preparation, due to the absence of wasteful counter-speculation, that is needed e.g., in first-price or Dutch auctions.

3 The Antisocial Attitude

In most multiagent applications it is assumed that the objective of an agent (or of a team of agents) is to maximize his absolute profit without caring for the profits made by the

[1] If the bidders don't have to estimate their private values, the dominant equilibrium exists independently of risk neutrality [11].

other agents. However, in many real-world applications it is more realistic to assume that agents may be present that try to gain as much money as possible *relative* to others (their competitors). In other words, in many scenarios it is wise to take into consideration the availability of "antisocial agents," that is, agents who accept small losses if they can inflict great losses to other agents. To make this more precise, we develop a formal description of this antisocial attitude. As a starting point for this formalization, it appears to be reasonable to assume that an antisocial agent wants to maximize the difference between his profit and the gain of his competitors; this means that the own profit on the one hand and the other agents' losses on the other hand are considered to be of equal importance from the point of view of this antisocial agent. In a two-player scenario, this view captures the antisocial agent's intention to be better than his rival. To achieve a higher degree of flexibility in describing and analyzing antisocial agents, it is useful to think of different degrees of anti-sociality like "aggressive anti-sociality" (where it is an agent's objective to harm competitors at any cost) and "moderate anti-sociality" (where an agent puts somewhat more emphasis on his own profit rather than the loss of other agents). These considerations lead to our formal specification of an antisocial agent (or an agent's antisocial attitude) as an agent who tries to maximize the weighted difference of his own profit and the profit of his competitors. Precisely, an antisocial agent i intends to maximize his *payoff*[2] that is given by

$$\mathit{payoff}_i = \qquad (1-d_i)\mathit{profit}_i \quad - \quad d_i \sum_{j \neq i} \mathit{profit}_j \quad , \tag{2}$$

where $d_i \in [0,1]$ is a parameter called *derogation rate*. The derogation rate is crucial because it formally captures, and allows to modify, an agent's degree of antisocial behavior. It is obvious that this formula covers "regular" agents by setting $d = 0$. If d is higher than 0.5, hurting others has greater priority than helping yourself. A purely destructive agent is defined by $d = 1$. We say an agent is *balanced antisocial* if $d = 0.5$, e.g., his own profit and the profit of his competitors are of equal importance. [3]
Please note that the described antisocial attitude, which is feasible in all kinds of competitive markets, *is* rational as the agents intend to benefit from their rivals' losses at a later date (e.g., by shortening a competitor's budget for future negotiations or by putting a rival completely out of business).

4 Antisociality and Vickrey Auctions

The implications of this formalized notion of an antisocial attitude on the Vickrey auction is enormous. In the remaining of this section, these implications are theoretically investigated. By combining equations 1 and 2 we get the (antisocial) payoff for agent A as a function of the bids b_a and b_b.

$$\mathit{payoff}_a(b_a, b_b) = \begin{cases} (1-d_a)(b_b - v_a) & \text{if } b_a \leq b_b \\ -d_a(b_a - v_b) & \text{if } b_a \geq b_b \end{cases} \tag{3}$$

[2] The payoff for a non-antisocial agent is simply his profit.

[3] When considering other types of auctions like uniform-price auctions, it might be wise to scale the sum in equation 2. In a Vickrey auction, however, all profits but one are zero which makes the scaling redundant.

Consider case vi) of Section 2. Agent A is not able to effectively win the auction, but the price agent B receives completely depends on A's bid. So, if A carefully adjusts his bid downwards, he is capable of reducing B's profit. Supposing that A knows B's private value v_b, his optimal strategy would be to bid $v_b + \epsilon$ (see Figure 1), which reduces B's profit to the absolute minimum of ϵ.

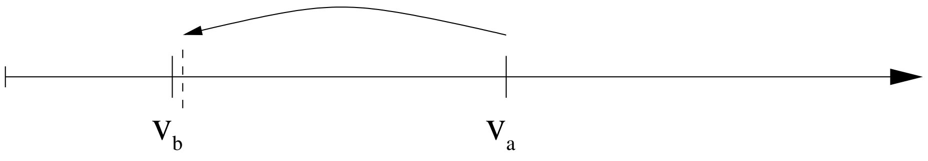


Fig. 1. A reduces B's profit to a minimum

However, if B knows A's prime costs v_a as well and he also prefers inflicting great losses to A over gaining small profits (i.e., his derogation rate is positive) he can bid $v_b + 2\epsilon$, rejecting a possible gain of ϵ and making A lose $v_a - v_b + 2\epsilon$.

As a result, if A's derogation rate d_a is 0.5, A should only bid $v_b + \frac{v_a - v_b}{2} + \epsilon$ to be safe from being overbid by B (see Figure 2). If B still tops A's bid, he is renouncing more money than A loses.

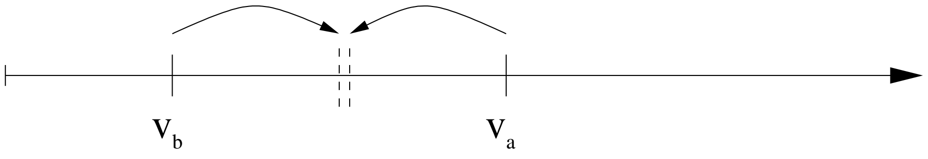


Fig. 2. Careful, aggressive bidding

If $d_b = 0.5$ as well, B's best strategy is to bid $v_b + \frac{v_a - v_b}{2}$.

Concluding, the following bidding strategy seems to be "safe" for an antisocial agent i (still under the unrealistic assumption that an agent knows private values of other agents: v_1 is the lowest private value, v_2 the second lowest):

$$b_i = \begin{cases} v_i - d_i(v_i - v_1) + \epsilon & \text{if } v_i > v_1 \\ v_i + d_i(v_2 - v_i) & \text{else} \end{cases} \tag{4}$$

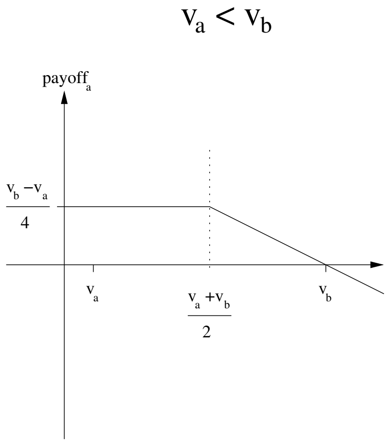
It is possible to omit the margin ϵ. We included it to avoid randomized experimental results that occur when two or more bidders share the same minimum bid. However, we will use the strict ϵ-less version in the theorems stated below.

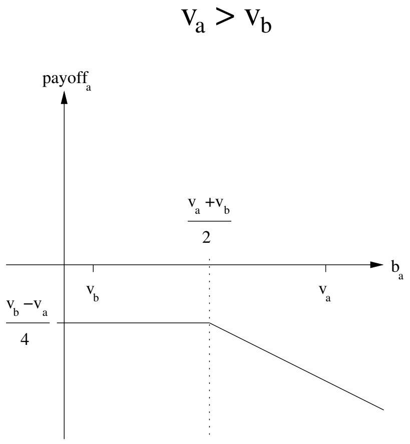
Theorem: In Vickrey auctions with balanced antisocial bidders ($\forall i : d_i = 0.5$), the strategy defined by equation 4 is in *Nash equilibrium*.

Proof: The assumption states that under the supposition that all agents apply this strategy, there is no reason for a single agent to deviate from it.
We consider the payoff of agent A. If v_a is the lowest private value, let B be the second lowest bidder. Otherwise, B represents the lowest bidder. It suffices to consider B as A cannot differentiate between the individual bidders and the Vickrey auction has a sole victor. There are only two possibilities for an antisocial agent to cause harm. The cheapest provider can hurt the second highest bidder and an inferior agent can reduce the profit of the lowest bidder. Assume B applies the antisocial strategy defined in 4.

$$b_b = \begin{cases} v_b - \frac{1}{2}(v_b - v_a) & \text{if } v_a \leq v_b \\ v_b + \frac{1}{2}(v_a - v_b) & \text{if } v_a \geq v_b \end{cases} = \frac{v_a + v_b}{2}$$

$$\Rightarrow \mathit{payoff}_a(b_a, b_b) = \mathit{payoff}_a\left(b_a, \frac{v_a + v_b}{2}\right) = \begin{cases} \frac{v_b - v_a}{4} & \text{if } b_a \leq \frac{v_a + v_b}{2} \\ -\frac{b_a - v_b}{2} & \text{if } b_a \geq \frac{v_a + v_b}{2} \end{cases}$$





$$\max_{b_a} \mathit{payoff}_a\left(b_a, \frac{v_a + v_b}{2}\right) = \frac{v_b - v_a}{4} \quad \Rightarrow b_a \leq \frac{v_a + v_b}{2}$$

Concluding, if A bids less than $\frac{v_a+v_b}{2}$, he only receives equal payoff; if he bids more, his payoff is diminishing. □

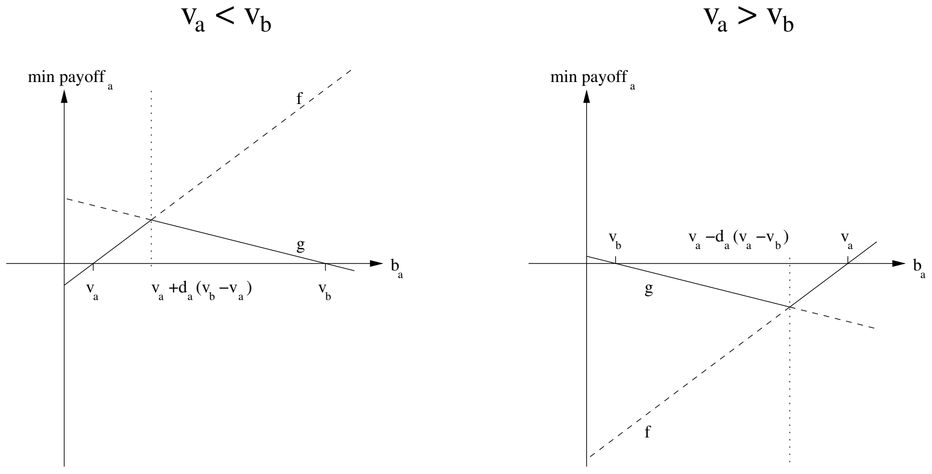
Theorem: The strategy defined by equation 4 is in *Maximin equilibrium* independently of the derogation rates of the bidders.

Proof: It is claimed that the strategy is an optimal strategy to reduce the possible losses that occur in worst case encounters. "Worst-case" means that the other bidders (represented by a single agent B again) try to reduce agent A's payoff as much as possible.

$$\min_{b_b} \mathit{payoff}_a(b_a, b_b) = \min_{b_b} \begin{pmatrix} (1-d_a)(b_b - v_a) & \text{if } b_b \geq b_a \\ -d_a(b_a - v_b) & \text{if } b_b \leq b_a \end{pmatrix}$$
$$= \min\{\underbrace{(1-d_a)(b_a - v_a)}_{f(b_a)}, \underbrace{-d_a(b_a - v_b)}_{g(b_a)}\}$$

f yields the minimum profit if A wins and g yields the minimum profit if he loses the auction. In the following, we consider the maximum of these minima.

$$\max_{b_a} \min_{b_b} \mathit{payoff}_a(b_a, b_b) = \max_{b_a} \min\{f(b_a), g(b_a)\}$$



Due to the fact that f is increasing and g is decreasing, the Maximin equilibrium point can be computed by setting $f(b_a) = g(b_a)$.

$$\begin{aligned} f(b_a) = g(b_a) &\Leftrightarrow (1-d_a)(b_a - v_a) = -d_a(b_a - v_b) \\ &\Leftrightarrow b_a - v_a - d_a b_a + d_a v_a = -d_a b_a + d_a v_b \\ &\Leftrightarrow b_a = v_a + d_a(v_b - v_a) \quad \square \end{aligned}$$

5 Antisocial Bidding in Repeated Auctions

On the basis of the theoretical foundations of the previous Section, we now develop an antisocial bidding strategy that can actually be used in realistic environments.

In the general case, an agent does not know the private value of other bidders. However, in principle an agent has several possibilities to figure out that value, for instance by means of *espionage*, careful *estimation* (e.g., by assuming a uniform distribution of private values), *colluding* with the auctioneer (e.g., bribing him) or by *learning* from previous auctions. This paper deals with the latter technique.

5.1 Revealing Private Values by Underbidding

We consider the auctioning of a fixed number of tasks, that repeats for several rounds. Now, suppose a balanced antisocial agent loses an auction in the first round. When the same task is auctioned for the second time, he bids zero. As a consequence, he wins the auction[4], and receives an amount equaling the second lowest bid, which is the private value of the cheapest agent (supposing this agent applied the dominant strategy). Thus, he is able to figure out the needed private value and can place his next bid right in the middle between the two private values. Using this technique, he loses the difference between both values once, but can safely cut off 50% of the competitor's profit for all following auction rounds. If the total number of rounds is high enough, the investment pays.

In a scenario where all other agents definitely follow the dominant bidding strategy and no counter-speculation is needed, an effective, antisocial bidding strategy looks like this:

1. Bid 0 (p=received price)
2. Bid $\begin{cases} v_i + d_i(p - v_i) & \text{if } v_i < p \\ v_i - d_i(v_i - p) + \epsilon & \text{else} \end{cases}$

Bidding zero is elegant but dangerous, especially if more than one agent is applying this strategy. In this case, one of the zero-bidding agents wins the auction, but is paid no money at all (because the second lowest bid is zero as well), thus producing a huge deficit.

5.2 Step by Step Approach

It's safer, but possibly not as efficient, to reduce a bid from round to round by a small margin s until the best agent's bid is reached. Figure 3 displays the modified strategy. If the step size s equals the private value ($s = v$), this algorithm emulates the aggressive zero-bidding strategy. The algorithm works somewhat stable in dynamic environments where agents can vanish and new ones appear from time to time. However, the strategy is not perfect, e.g., if two balanced antisocial agents apply this strategy, the more expensive agent is only able to reduce the winning agent's profit by 25% because he is usually not able to figure out the real private value of the cheaper agent in time. Nevertheless, this is still tolerable, because in a group of antisocial agents a bidder never takes *more* risk than his derogation rate prescribes.

Generally, a careful agent should use a small step size s in order to be safe that the competitor already suffered huge losses before he makes negative profit himself. A reasonable setting of s depends on the number of rounds, the distribution of private values and his derogation rate (an upper bound for s is specified in [1]).

5.3 Leveled Commitment Contracting

If the task execution contracts are not binding and can be breached by paying a penalty (leveled commitment contracting [15,16,3]), the unavoidable loss an agent produces by

[4] unless some other agent bids zero as well.

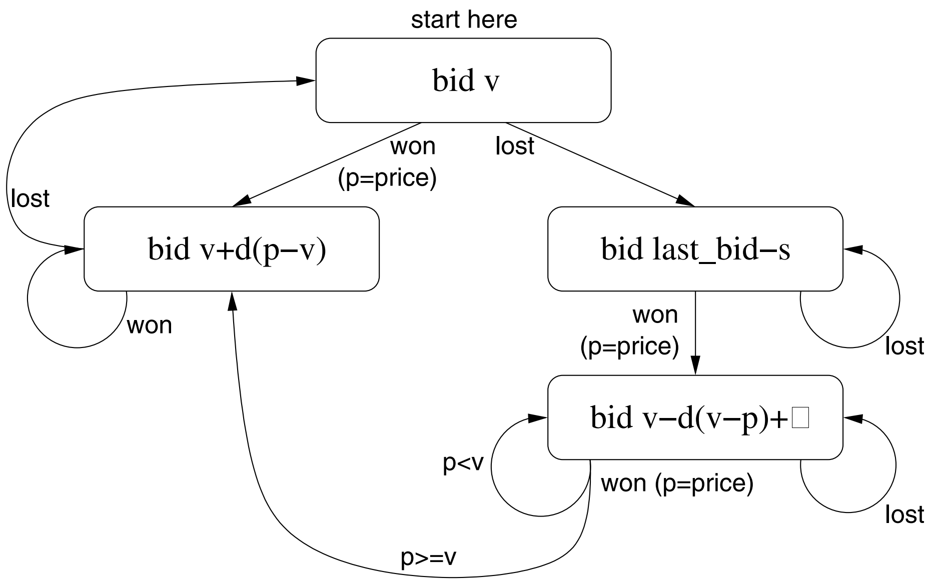


Fig. 3. Antisocial strategy for repeated Vickrey auctions

underbidding the cheapest competitor can be reduced by breaking the negative contract. Due to the fact that the only reason for closing that deal is to figure out the private value of another agent, the agent has no incentive to really accomplish the task. Therefore, a contractee will break the contract if the loss he makes by accepting the contract is greater than the penalty he pays by breaking the deal. Supposing the common definition of a penalty as a fraction of the contract value, agent i is better off breaching the contract if

$$p \leq \frac{v_i}{pr + 1} \tag{5}$$

with p being the actual task price and $pr \in [0; 1]$ the penalty rate. To give an example, under the assumption that $pr = 0.25$, an agent should break a contract if the task price is less or equal than $\frac{4}{5}$ of his private value. When the distribution of prime costs is uniform, this is true in 80% of all possible cases.

6 Experimental Results

The experimental setting investigated in this paper is similar to the one used in [3]. Due to reasons of limited space, we can only present some of the experiments we conducted. Please see [1] for more detailed results including a randomized cost table.

There is a number of buyers or *contractees* (CE_i) who are willing to execute tasks. Contractees associate prime costs with task execution and are interested in tasks whose prices are higher than their own costs. All prices and bids are integer values ($\epsilon = 1$).

Whenever the selling of a task is announced, each interested contractee calculates and submits one sealed bid. The contractee who submitted the lowest bid is declared as the winner of the auction, and the *second lowest* bid is taken as the price of the announced task; the contractee is paid this price and executes the task. If there are two or more equal winning bids, the winner is picked randomly. As a contractee wants to earn money for handling tasks, his private value of a task is his prime costs plus ϵ. It is assumed that each contractee can execute as many tasks as he wants during one round (full commitment contracting).

Table 1. Fair cost table

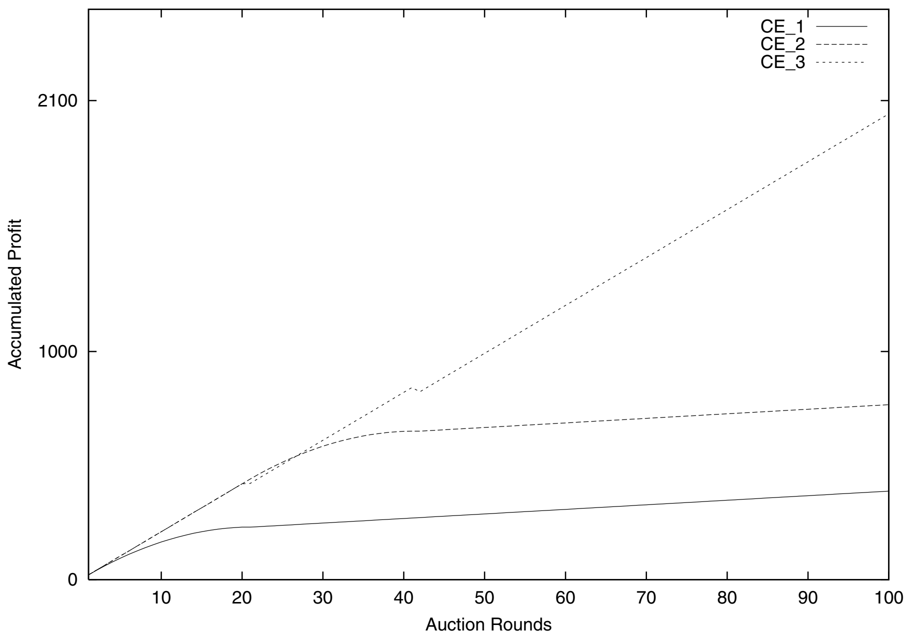
	Task 1	Task 2	Task 3
CE_1	70	50	30
CE_2	50	30	70
CE_3	30	70	50

Table 1 contains the prime costs of three contractees with exactly identical abilities. Each contractee has one task, that he can handle for the cheapest price. If all three truly bid their private values for 100 rounds, each one would gain $((50 - 30) + 1) \times 100 = 2100$. Figure 4 shows the profits accumulated by the contractees in 100 rounds. CE_1 and CE_2 apply the dominant strategy and bid their prime costs plus one. CE_3, however, is antisocial and tries to harm his competitors by reducing their profits to a minimum. As CE_3 is the only antisocial agent and because his derogation rate is 1, he could use a very large step size, e.g., $s_3 = v_3$. We chose a careful step size setting ($s_3 = \epsilon$) for two reasons. First of all, CE_3 may not know he is the only antisocial bidder, and secondly, this setting superiorly visualizes how the antisocial strategy works.

Despite the dominant strategy, CE_1 and CE_2 are incapable of gaining their regular profit (2100). CE_3 outperforms his rivals by losing only 60. The summed up profit of the entire group of contractees is reduced by more than 50% by a single agent using an aggressive strategy. This emphasizes the particular vulnerability of Vickrey auctions to "irrational" bidding. There is no counter-strategy available for CE_1 or CE_2. As they apply a dominant strategy, they already make the highest possible payoff (which is almost nil) according to their definition of payoff. However, if they decided to become antisocial as well, they would be able to increase their payoff. In certain scenarios, this might lead to an antisocial chain reaction.

It might appear confusing at the first glance that an agent who does not care for his own profit at all ($d_3 = 1$) nevertheless makes the highest profit. This odd effect can be explained by the peaceableness of the fellow bidders and the ϵ that we added in equation 4. CE_3 *risks* his entire profit in order to hurt CE_1 and CE_2, but as both are completely harmless, he keeps his gain.

If all three contractees are antisocial, overall performance breaks down as expected (see Figure 5). The agents cut off more than a quarter of profits of their rivals due to a mutual derogation rate of 0.5.



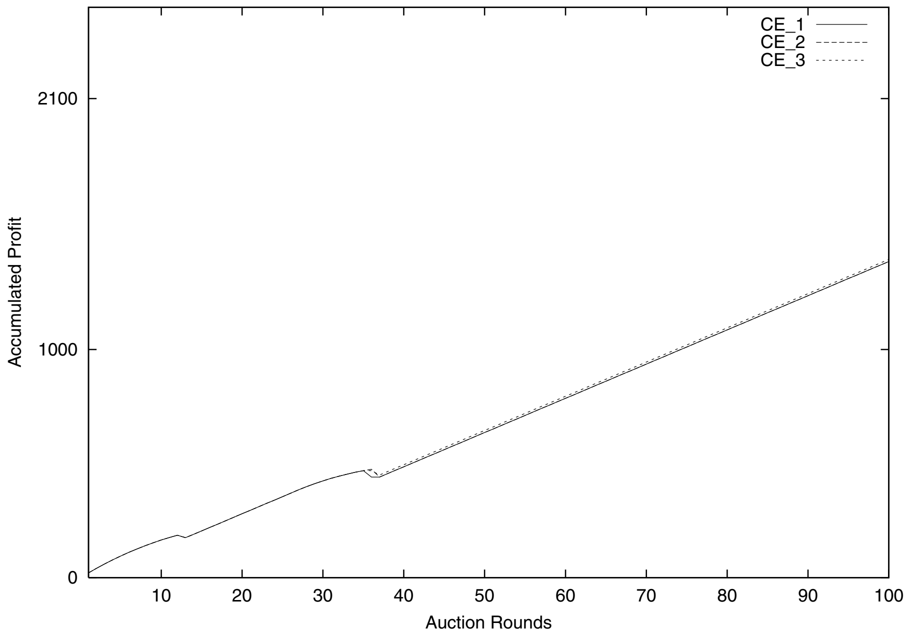
CE_3 is antisocial ($d_3 = 1, s_3 = \epsilon$).

Fig. 4. Identical contractees (1/3 antisocial)

7 Conclusions

The antisocial attitude for agents and its formalization introduced in this paper leads to a significant need for important changes in strategic behavior of agents. As argued above, and as it is obviously implied by many real-world applications, it is necessary to take the existence of antisocial agents into consideration. The paper focused on the Vickrey auction and showed that this auction protocol, in addition to other known deficiencies [14,13,12], is vulnerable to antisocial bidders. This is an effect of the second-price policy which enables easy price manipulation and emphasizes the required confidentiality of private values and the need for a secure auction protocol that hides those values from all parties, including the auctioneer [2]. As the common *English* auction for private value bidders is equivalent to the Vickrey auction [9,11], all strategies in this paper work for English auctions as well. The inability to prevent profit reduction can be regarded as a major disadvantage of those two auction types as *Dutch* and *first-price sealed-bid* auctions do not suffer from antisocial strategies.

One problem that arises using the new strategy in repeated Vickrey auctions is that if there is more than one inferior, antisocial agent, it would be desirable if only the one that intends to cut off the cheapest agent's profit by the highest margin reduces his bid. All other bidders should stay with bidding their private value, since they would lose money once without harming anyone in the following rounds. This would require the antisocial agents to make a special arrangement, which may not be feasible in certain settings. The



All contractees are antisocial ($d_i = 0.5$, $s_i = \epsilon$).

Fig. 5. Identical contractees (3/3 antisocial)

behavior described in this paper can be seen as an opposite of *bidder collusion* where bidders coordinate their bids in order to help each other (harming the auctioneer). In contrast, antisocial agents bid with the intention to harm fellow bidders.

As a next research step we want to explore “anti-sociality” and strategies for antisocial agents in more detail. To do so, we plan to examine these strategies in more complex environments where the number of contractees varies over time and tasks can be recontracted. Furthermore, we intend to investigate types of collaboration of antisocial agents that allow them to act more efficiently.

References

1. F. Brandt. Antisocial Bidding in Repeated Vickrey Auctions. Technical Report FKI-241-00, Institut für Informatik, Technische Universität München, 2000.
2. F. Brandt. Cryptographic protocols for secure second-price auctions. In M. Klusch and F. Zambonelli, editors, *Cooperative Information Agents V*, volume 2182 of *Lecture Notes in Artificial Intelligence*, pages 154–165, Berlin et al., 2001. Springer.
3. F. Brandt, W. Brauer, and G. Weiß. Task assignment in multiagent systems based on Vickrey-type auctioning and leveled commitment contracting. In M. Klusch and L. Kerschberg, editors, *Cooperative Information Agents IV*, volume 1860 of *Lecture Notes in Artificial Intelligence*, pages 95–106, Berlin et al., 2000. Springer.

4. K. Danielsen and M. Weiss. User control modes and IP allocation. http://www.press.umich.edu/jep/works/DanieContr.html, 1995. presented at MIT Workshop on Internet Economics.
5. K.E. Drexler and M.S. Miller. Incentive engineering for computational resource management. In B.A. Huberman, editor, *The Ecology of Computation*. The Netherlands, 1988.
6. B. Huberman and S.H. Clearwater. A multiagent system for controlling building environments. In *Proceedings of the 1st International Conference on Multiagent Systems (ICMAS-95)*, pages 171–176, Menlo Park, CA, 1995. AAAI Press.
7. P. Jehiel, B. Moldovanu, and E. Stacchetti. How (not) to sell nuclear weapons. *American Economic Review*, 86:814–829, 1996.
8. J.K. MacKie-Mason and H.R. Varian. Pricing the internet. In *Proceedings of the Public Access to the Internet Conference*. JFK School of Government, 1993.
9. R.P. McAfee and J. McMillan. Auctions and Bidding. *Journal of Economic Literature*, 25:699–738, 1987.
10. P.R. Milgrom and R.J. Weber. A Theory of Auctions and Competitive Bidding. *Econometrica*, 50:1089–1122, 1982.
11. E. Rasmusen. *Games and Information*. Basil Blackwell, 1995.
12. M.H. Rothkopf and R.M. Harstad. Two models of bid-taker cheating in Vickrey auctions. *Journal of Business*, 68(2):257–267, 1995.
13. M.H. Rothkopf, T.J. Teisberg, and E.P. Kahn. Why are Vickrey auctions rare? *Journal of Political Economy*, 98(1):94–109, 1990.
14. T.W. Sandholm. Limitations of the Vickrey auction in computational multiagent systems. In *Proceedings of the 2nd International Conference on Multiagent Systems (ICMAS-96)*, Menlo Park, CA, 1996. AAAI Press.
15. T.W. Sandholm and V.R. Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 328–335, 1995.
16. T.W. Sandholm and V.R. Lesser. Advantages of a leveled commitment contracting protocol. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 126–133, 1996.
17. Sun Microsystems. Webmart: Managing shared resources by market mechanisms. http://www.sun.com:80/960201/cover/webmart.html, 1996.
18. W. Vickrey. Counter speculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.
19. C.A. Waldspurger, T. Hogg, B. Huberman, J.O. Kephart, and W.S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, 1992.
20. G. Weiß, editor. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, MA, 1999.

Formalizing a Language for Institutions and Norms

Marc Esteva[1], Julian Padget[2], and Carles Sierra[1]

[1] Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC
08193 Bellaterra, Barcelona, Spain.
tel:+34-93-5809570
{marc, sierra}@iiia.csic.es
[2] Department of Computer Science
University of Bath, BATH BA2 7AY, UK
tel:+44-1225-826971
jap@cs.bath.ac.uk

Abstract. One source of trust for physical trading systems is their physical assets and simply their presence. A similar baseline does not exist for electronic trading systems, but one way in which it may be possible to create that initial trust is through the abstract notion of an institution, defined in terms of norms [19] and the scenes within which (software) agents may play roles in different trading activities, governed by those norms. We present here a case for institutions in electronic trading, a specification language for institutions (covering norms, performative structure, scenes, roles, etc.) and its semantics and how this may be mapped into formal languages such as process algebra and various forms of logic, so that there is a framework within which norms can be stated and proven.

1 Introduction

Human interaction very often follows conventions, that is, general agreements on language, meaning, and behaviour. By following conventions humans decrease uncertainties about the behaviour of others, reduce conflicts of meaning, create expectations about the outcome of the interaction and simplify the decision process by restricting to a limited set the potential actions that may be taken. These benefits explain why conventions have been so widely used in many aspects of human interaction: trade, law, games, etc.

On some occasions, conventions become foundational and, more importantly, some of them become norms. They establish how interactions of a certain sort will and must be structured within an organization. These conventions, or norms, become therefore the essence of what is understood as human institutions [19]. This is so for instance in the case of auction houses, courts, parliaments or the stock exchange. Human institutions not only structure human interactions but also enforce individual and social behaviour by obliging everybody to act according to the norms.

The benefits obtained in human organizations by following conventions become even more apparent when we move into an electronic world where human interactions are mediated by computer programs, or agents. Conventions seem necessary to avoid conflicts in meaning, to structure interaction protocols, and to limit the action repertoire in a setting where the acting components, the agents, are endowed with limited rationality. The notion of electronic institution becomes thus a natural extension of human institutions by permitting not only humans but also autonomous agents to interact with one another.

Considering the computer realization of an institution, we have the view that *all* interactions among agents are realized by means of *message interchanges*. Thus, we take a strong dialogical stance in the sense that we understand a multi-agent system as a type of *dialogical system*. The interaction between agents within an institution becomes an illocution exchange. In accordance with the classic understanding of illocutions (e.g. [2] or [23]), illocutions "change the world" by establishing or modifying the *commitments* or *obligations* of the participants. Therefore, formally speaking, an agent will be in this context any entity capable of establishing commitments. This notion becomes the cornerstone of the construction of institutions because otherwise no enforcement or penalty could ever be applied. In a sense, institutions exist because they are the warrants of the satisfaction of the commitments established by the participants.

In this paper we focus on the specification and potential animation of electronic institutions, and specifically on the formal modelling of agent interactions in the framework of electronic institutions. To do so, we explore the application of process algebraic models, using the Ambient calculus [6] (some history of process algebra, justification and citation of earlier work omitted here due to space limitations).

This paper is structured as follows. In section 2 we introduce the background and motivation for our work in electronic institutions, from which we abstracted the components and principles that constitute the institutional description language discussed in section 3, and which we subsequently use to express a modelling of the original Fish-Market. Then, in section 4 we sketch a formal basis for the language in the Ambient Calculus and conclude (section 5) with a summary of related work.

2 A Model Institution

Much of our work since 1995 has taken as a reference point the physical market for auctioning fish in the town of Blanes on the Costa Brava. From this physical model we have abstracted what we call *scenes*, for the admission of buyers, the admission of sellers, the auction room (where a standard downward bidding/Dutch auction format is employed), buyers' settlement, sellers' settlement and a back-room accommodating the accountant, quality assessor and other institutional functionaries. Thus, for each activity that can take place in the institution, there is a corresponding scene, in which interactions between agents are articulated through agent group meetings that follow a well-defined communication protocol—in fact, in our institutions, agents may *only* interact within the context of a scene. This has been described and discussed in detail in [17], while a more general, but also more technical approach appears in [22]. This set of scenes and the connections between them—what roles agents may play in them, how many of each role, to which scenes they may move—constitute the *performative structure* for the electronic institution (see Figure 1). The diagram, as we will explain in subsection 3.2, presents a simplified version of the Fish Market performative structure. The purpose of this diagram is to show the different scenes which comprise the institution by means of

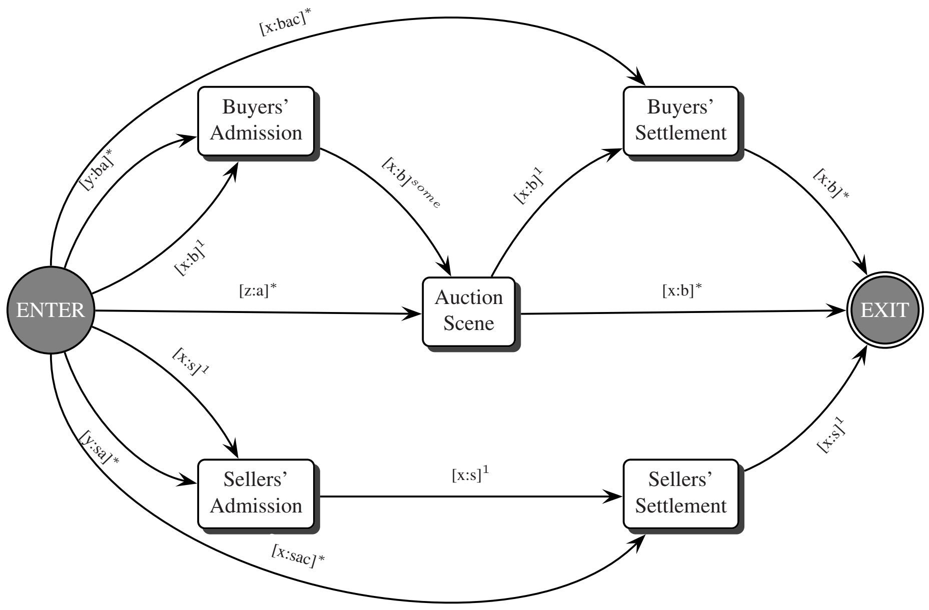
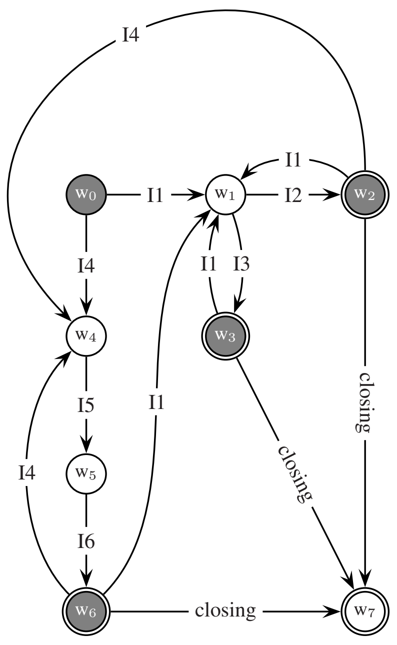


Fig. 1. Performative structure of the Fish Market

a transition graph. Thus, the black circle on the left hand side denotes the start scene and that on the right hand side surrounded by a line is the end scene. In between, there are scenes (rectangles with rounded corners) and arcs connecting them. The arcs are labelled with variable:role pairs, where a = auctioneer, b = buyer, ba = buyers' admitter, bac = buyers' accountant, s = seller, sa = sellers' admitter and sac = sellers' accountant. The superscript on the labels indicates whether agents following the arc will start a new scene execution or whether they will join one, some or all active executions of the target scene. A * denotes that a new execution will start, a 1 denotes that agents will join just one execution of the target scene, a *some* denotes that agents will join a subset of the executions of the target scene and *all* denotes that agents will join all the executions of the target scene. For instance, there can be multiple instances of the auction scene, each one initiated by a different auctioneer. Then a buyer can choose to join some of them. It is helpful to know that the system is intended to be initialized by injecting the (staff) agents via the enter node, whence they traverse the performative structure to reach their allotted scenes. Subsequently, buyers and sellers will also enter the market via the enter node and follow the paths assigned to the roles they have adopted.

Within each scene, we use a transition graph labelled with illocutions to define the structure of a conversation and to identify the states at which an agent may join or leave the scene and which agents may say which illocutions (see the Buyers' Settlement scene in Figure 2). The purpose of this diagram is to formalize what an agent may say, which agents may say what, in what order things may be said and at what points a conversation may begin and end (denoted by the access and exit nodes). Each arc is labelled with an illocution, which comprises a particle (in this case, request, accept, deny, inform, or pay), a sender (a variable/role pair), a receiver (idem) and some content. The role



Where

● a denotes an access node

◎ b denotes an exit node

◉ c denotes an access/exit node

and the illocutions are as follows:

```
I1 (request (?x b) (?y bac)
    (update-credit ?value))
I2 (accept (!y bac) (!x b)
    (accept ‘‘update-credit"))
I3 (deny (!y bac) (!x b) (deny
    ?deny-code))
I4 (request (?x b) (?y bac)
    statement)
I5 (inform (!y bac) (!x b)
    (statement (?g1 ?p1) ...
    (?gn ?pn)))
I6 (pay (!x b) (!y bac)
    (payment ?value))
```

Fig. 2. Conversation graph and illocutions for Buyers' Settlement

identifiers are the same as those given for the performative structure of Figure 1. A fuller explanation of the language of illocutions appears in the next section.

A novel feature of our design is the use of so-called *governor* agents, which mediate between external agents and the institution. These serve several purposes: (i) the governors move around the scenes of the institution on behalf of their external agents because neither do we want to give an external agent potential access to institution internals, nor do we expect that any agent would normally want to put itself in such a position (ii) they may be able to answer questions posed by the external agent about the institution (iii) they shall ensure adherence to the performative structure and the conversation protocol (iv) they may communicate with other governor agents regarding the running of the institution.

3 A Language for Institutions

We have developed a simple declarative language for specifying the components of our electronic institutions. This language reflects what we have concluded are the key elements in the specification, namely the institution itself, its *performative structure*, the *scenes* making up the performative structure and the *normative rules*. An informal specification of the description language follows here and for the remainder of this section we examine its use in the specification of the (infamous) Fish Market. A complete and

A dialogic framework specification

dialogic-framework: a *name* with which to refer to this framework.
ontology: a *name* referring to a defined ontology (q.v.).
content-language: a *name* defining the content language. It has to be KIF[13], PROLOG or LISP. The intention is to allow for the encoding of the knowledge to be exchanged among agents using the vocabulary offered by the ontology. These propositions are then embedded in the language in accordance with speech act theory [23], by means of the *illocutionary particles*.
illocutionary-particles: a *list* of *names* of illocutionary particles to be used in the illocutions.
external-roles: a *list* of *names* of roles that external agents may play.
internal-roles: a *list* of *names* of roles that internal (staff) agents may play.
social-structure: a *list* of *triples* of two role *names* and the *name* of relationship between them.

Fig. 3. Elements of a dialogic framework

```
(define-dialogic-framework
  fm-dialogic-framework as
  ontology = fm-ontology
  content-language = PROLOG
  illocutionary-particles = (request accept deny inform commit pay)
  external-roles = (buyer seller)
  internal-roles = (boss buyer-admitter seller-admitter
                    auctioneer buyer-accountant seller-acountant)
  social-structure =((boss < buyer-admitter) (boss < seller-admitter)
                    (boss < auctioneer) (boss < buyer-accountant)
                    (boss < seller-accountant) (buyer incompatible seller))
)

(define-dialogic-framework
  buyer-settlement-df as
  ontology = buyer-settlement-ontology
  content-language = PROLOG
  illocutionary-particles = (request inform accept deny pay)
)
```

Fig. 4. The FishMarket and Buyer settlement scene dialogic frameworks

detailed description of the language, called ISLANDER, can be found in [11]. In next subsections we present an informal static semantics for each of the components.

3.1 Ontologic and Communicational Components: The Dialogic Framework

The dialogic framework [18], determines the illocutions that can be exchanged between the participants. In order to do so, an ontology is defined that fixes what are the possible values for the concepts in a given domain, e.g goods, participants, roles, locations, time intervals, etc. The elements of the dialogic framework are summarized in Figure 3.

We consider that the communication language *expressions* are constructed as formulae of the type $(\iota\,(\alpha_i : \rho_i)\,(\alpha_j : \rho_j)\,\varphi\ \tau)$ where ι is an *illocutionary particle*, α_i and α_j are terms which can be either agent variables or agent identifiers, ρ_i and ρ_j are terms which can be either role variables or role identifiers, φ is an expression in the *content language* and τ is a term which can be either a time variable or a time constant.

Two examples of dialogic frameworks appear in Figure 4: (i) for the FishMarket as a whole, noting in particular the illouctionary particles that may be used to tag an illocution, the roles, the role hierarchy and the declaration that an agent may not be a buyer and a seller at the same time. (ii) for the buyer settlement scene, where the particles are restricted to those stated, but the other aspects are inherited from the institutional dialogic framework.

3.2 Social Components: Scenes and Performative Structure

We begin by explaining precisely what, for our needs, we regard to be the purpose of a scene. The activities in an electronic institution are the composition of multiple, distinct, possibly concurrent, dialogic activities, each one involving different groups of agents playing different roles. For each activity, interactions between agents are articulated through agent group meetings, which we call *scenes*, that follow well-defined communication protocols.

A scene protocol is specified by a directed graph where the nodes represent the different states of the conversation and the directed arcs are labelled with illocution schemes or timeouts. The graph has a single initial state (unreachable once left) and a set of final states representing the different endings of the conversation. There is no arc connecting a final state to some other state.

Normally, the correct evolution of a conversation protocol requires a certain number of agents to be present for each role participating in the protocol. Thus, we define a minimum and a maximum number of agents for each role and this constraint is enforced by the institution. However, because we aim at modelling multi-agent conversations whose set of participants may dynamically vary, scenes allow for agents either joining or leaving at particular points during a conversation. We specify this by identifying, for each role, a set of access and exit states. The final state(s) appear in the exit states of each role, to permit all the agents to leave when the scene is finished. The initial state must be in the set of access states for those roles whose minimum is greater than zero, since this constraint implies that an agent playing that role is required in order to start the scene.

The notion of performative structure is perhaps the most complex since it models the relationships among scenes. In particular, we wish to highlight that although conversations (scenes) are quite widely viewed as the unit of communication between agents, limited work has been done concerning the modelling of the relationships among different scenes. This issue arises when these conversations are embedded in a broader context, such as, for instance, organizations and institutions. If this is the case, it does make sense to capture the relationships among scenes. Thus, while a scene models a particular multi-agent dialogic activity, more complex activities can be specified by establishing relationships among scenes.

In general, the activity represented by a performative structure can be depicted as a collection of multiple, concurrent scenes. Agents navigate from scene to scene constrained by the rules defining the relationships among scenes. Moreover, the very same agent can be possibly participating in multiple scenes at the same time. Likewise, there

A scene specification

roles: a *list* of *names* of role that may participate in the scene.
dialogic-framework: the *name* of the dialogic framework to be used for communication within the scene.
states: a *list* of the *names* of the states of the conversation graph.
initial-state: a *name* identifying the initial state.
final-states: a *list* of *name*s identifying final states.
access-states: a *list* of *pairs* of role *name* and a *list* of states, identifying which roles may join at which states.
exit-states: a *list* of *pairs* of role *name* and a *list* of states, identifying which roles may leave at which states.
agents-per-role: a *list* of *triples* of role *name*, minimum *integer* and maximum *integer*, defining the constraints on the population of a particular role.
connections: a *list* of the transitions between scene states. Each one comprises a preceding state *name*, a succeeding state *name*, and either an *illocution-scheme* with some constraints over scenes variables which must be satisfied to progress through this transition or a timeout that will trigger the transition when will expire.

Fig. 5. Elements of a scene specification

```
(define-scene
  buyer-settlement-scene as
  roles = (buyer buyer-accountant)
  scene-dialogic-framework =  buyer-settlement-df
  states = (w0 w1 w2 w3 w4 w5 w6 w7)
  initial-state = w0
  final-states = (w7)
  access-states = ((buyer (w0 w2 w3 w6))
                   (buyer-accountant (w0)) )
  exit-states = ((buyer (w2 w3 w6 w7)) (buyer-accountant (w7)))
  agents-per-role = ((0 <= buyer <= 1)
                     (1 <= buyer-accountant <= 1) )
  connections = ((w0 w1 buyer-settlement-i1)
                 (w1 w2 buyer-settlement-i2)
                 (w1 w3 buyer-settlement-i3)
                 (w0 w4 buyer-settlement-i4)
                 (w4 w5 buyer-settlement-i5)
                 (w5 w6 buyer-settlement-i6)
                 (w2 w1 buyer-settlement-i1)
                 (w3 w1 buyer-settlement-i1)
                 (w6 w1 buyer-settlement-i1)
                 (w2 w4 buyer-settlement-i4)
                 (w3 w4 buyer-settlement-i4)
                 (w6 w4 buyer-settlement-i4)
                 (w2 w7 closing)
                 (w3 w7 closing)
                 (w6 w7 closing)
                )
)
```

Fig. 6. FishMarket buyer settlement scene

may be multiple concurrent instantiations of a scene, so we must also consider whether the agents following the arcs from one scene to another are allowed to start a new scene execution, whether they can choose to join just one or a subset of the active scenes, or even join all the active scenes. Furthermore, we may associate constraints with each arc, such that and agent must satisfy the constraint in order to traverse the arc. In order to capture the relationship between scenes we use a special type of scenes the so-called transitions. The type of a transition allows to express agents synchronization, choose points where agents can decide which path to follow or parallelisation points where agents are sent to more than one scene. Transitions can be seen as a type of routers in the context of a performative structure.

From a structural point of view, performative structures' specifications must be regarded as networks of scenes. The connections among the scenes defines which agents depending on their role can move from one scene to other(s) through the defined transitions. In other words, the performative structure defines which scenes can be reached by each one of the different roles.

In Figure 6 we give a concrete example of a scene specification, that shows the buyer settlement scene (thus corresponding to the pictorial presentation of the conversation graph in Figure 2). The dialogic framework for this scene was given earlier (Figure 4), while the remainder of the scene specification is effectively a textual description of the conversation graph complete with illocution labels.

At a higher level, Figure 8 shows the declaration of the scenes comprising the Fish-Market and the relationships between them. In effect, this is the textual counterpart to the diagram in Figure 1. In the diagram transitions are omitted for simplicity because there are no synchronisation or choice points, and only one agent can progress at each time through each one of the arcs. Also the connections that will allow staff agents to leave are omitted. This is, a path that connects each scene with the exit scene labelled with the role of the satff agent in charge of it. The textual specification also expresses some constraints not present in the diagram, such as limits on the number of instances of a particular scene and conditions upon connections (see for example, the connection from auction room to transition t9, which has a condition stating that an agent may only take this arc if it has not acquired the obligation to make a payment).

3.3 Normative Rules and Institutions

The norms which govern an organization are one of the key sources of trust for potential participants, since they define the commitments, obligations and rights of participating agents. As described so far, the performative structure constrains the behaviour of participating agents at two levels:

1. *intra-scene:* Scene protocols dictate for each agent role within a scene what can be said, by whom, to whom, and when.
2. *inter-scene:* The connections between the scenes of a performative structure define the possible paths that agents may follow depending on their roles. Furthermore, the constraints over output arcs impose additional restrictions on agents attempting to reach a target scene.

These norms are, in effect, local. But, it is the agent's actions *within* a scene that may have consequences that either limit or expand its acting possibilities in subsequent

A performative structure specification

scenes: a *list* comprising a *name* for the scene, the *class* of the scene. If there can be multiple instantiations of a scene, this will be denoted by the word 'list' after the class name.
transitions: a *list* comprising a *name* for the transition and the *class* of the transition.
connections: a *list* containing the connections from scenes to transitions and from transitions to scenes. In the first case the connection is expressed by the source scene *name*, the target transition *name*, a *list* of *pairs* of *agent-variable* and role *name*, and a *list* of constraints that will restrict agents movements. In the second case is expressed by the source transition *name*, the target scene *name*, a *list* of *pairs* of *agent-variable* and role *name*, and a *name* defining if a new execution of the scene will be created or if the agent(s) will go to one, some or all current executions.
initial-scene: the *name* of the initial scene—from one of those given in **scenes**.
final-scene: the *name* of the final scene—from one of those given in **scenes**.

Fig. 7. Elements of the performative structure

```
(define-performative-structure
  fm-performative-structure as
  scenes = ((enter root-scene) (exit output-scene)
            (buyer-admission buyer-admission-scene)
            (seller-admission seller-admission-scene)
            (auction-room auction-room-scene list)
            (buyer-settlement buyer-settlement-scene)
            (seller-settlement seller-settlement-scene))
  transitions = ((t1 AND-AND)  (t2 AND-AND)  (t3 AND-AND)
                 (t4 AND-AND)  (t5 AND-AND)  (t6 AND-AND)
                 (t7 AND-AND)  (t8 AND-AND)  (t9 AND-AND)
                 (t10 AND-AND) (t11 AND-AND) (t12 AND-AND) (t13 AND-AND) )
  connections =
    ((enter t1 ((x buyer-admitter)))
     (t1 buyer-admission ((x buyer-admitter)) new)
     (enter t2 ((x buyer)))
     (t2 buyer-admission ((x buyer)) one)
     (enter t3 ((x seller-admitter)))
     (t3 seller-admission ((x seller-admitter)) new)
     (enter t4 ((x seller)))
     (t4 seller-admission ((x seller)) one)
     (enter t5 ((x auctioneer)))
     (t5 auction-room ((x auctioneer)) new)
     (enter t6 ((x buyer-accountant)))
     (t6 buyer-settlement ((x buyer-accountant)) new)
     (enter t7 ((x seller-accountant)))
     (t7 seller-settlement ((x seller-accountant)) new)
     (buyer-admission t8 ((x buyer)))
     (t8 auction-room ((x buyer)) some)
     (auction-room t9 ((x buyer)) (not obligued(x,pay,buyer-settlement)))
     (t9 exit ((x buyer)) one)
     (auction-room t10 ((x buyer)))
     (t10 buyer-settlement ((x buyer)) one)
     (buyer-settlement t11 ((x buyer)))
     (t11 exit ((x buyer)) one)
     (seller-admission t12 ((x seller)))
     (t12 seller-settlement ((x seller)) one)
     (seller-settlement t13 ((x seller)))
     (t13 exit ((x seller)) one) )
   initial-scene = enter
   final-scene = exit
)
```

Fig. 8. Performative structure for the FishMarket

scenes. The consequences we have identified take two different forms. Some actions create commitments for future actions, which may be interpreted as obligations. Other actions may affect the paths an agent may take through the performative structure because it may change which constraints are satisfied. Both types of consequences will need to be observed and maintained by an institution on a per agent basis.

For instance, a trading agent winning a bidding round within an auction house is obliged subsequently to pay for the acquired good. Considering the performative structure in Figure 1 that implies that the trading agent has to move at some time to the buyers' settlements scene to pay for the acquired good. Notice that although the auction scene is connected to the output scene, the path is disallowed to agents unless they fulfill their pending payments. From this example, we can deduce that norms must define the actions that will provoke the activation of the norm, the obligations that agents will have and the actions that agents must carry out in order to fulfill the obligations.

As we are specifying dialogical institutions, agents actions are expressed as a pair of illocution scheme and scene where it is uttered. We need both components because the same illocution could appear in more than one scene. The scene gives the context in which the illocution must be interpreted and of course, this affects the consequences that the utterance of the illocution has. That is to say, the same illocution may have different consequences in different scenes because it is uttered in a different context.

As we have said some of the terms of an illocution scheme are variables. The activation of a norm may depend on the values of these variables in the uttered illuctions. For instance, we specify a norm that says that if a buyer submits a bid which exceeds his credit limit, then the auctioneer is obliged to sanction him (see the `sanction` norm in Figure 10). Each time a buyer submits a bid the value of his bid and the level of his current credit determines whether the norm is activated and whether the auctioneer has to sanction him. These restrictions are specified as boolean expressions over illocution scheme variables and a norm will not be activated if they are not satisfied.

In order to represent the deontic notion of obligation (see [26] for background details) we set out the predicate *Obl* as follows:

$$\mathit{Obl}(x, \psi, s) = \text{agent } x \text{ is obliged to do } \psi \text{ in scene } s.$$

where ψ is taken to be an illocution scheme. We denote the set of obligations by *Obl* and any concrete obligation by $\mathit{obl}_i \in \mathit{Obl}$. Next we introduce some specific rules, the so-called *normative rules*, to capture which agent actions (illocutions) have consequences that need to be recorded. Given a performative structure and a metalanguage, the normative rules will have the following schema:

$$\begin{array}{rl} (s_1, \gamma_1) \wedge \ldots \wedge (s_m, \gamma_m) \wedge & \text{LHS part 1: scene/illocution scheme pairs} \\ e_1 \wedge \ldots \wedge e_n \wedge & \text{LHS part 2: } \begin{cases} \text{boolean expressions over} \\ \text{illocution scheme variables} \end{cases} \\ \neg(s_{m+1}, \gamma_{m+1}) \wedge \ldots \wedge \neg(s_{m+n}, \gamma_{m+n}) & \text{LHS part 3: } \begin{cases} \text{negated scene/illocution} \\ \text{scheme pairs} \end{cases} \\ \to \mathit{obl}_1 \wedge \ldots \wedge \mathit{obl}_p & \text{RHS: obligations that hold} \end{array}$$

where $\neg$ marks a defeasible negation. The meaning of these rules is that if the illocutions $(s_1, \gamma_1), \ldots, (s_m, \gamma_m)$ have been uttered, the expressions $e_1, \ldots, e_n$ are satisfied and

A norm specification

antecedent: a *list* comprising an arbitrary number of *pairs* of scene *name* and illocution-scheme *name* and a *list* of boolean expressions over illocution scheme variables.
defeasible-antecedent: a *list* comprising an arbitrary number of *pairs* of scene *name* and illocution-scheme *name*.
consequent: a *list* of obl *predicate*s.

Fig. 9. Elements of a norm specification

```
(define-norm obligation2pay as
  antecedent =
  ((auction-room
    (inform (?y auctioneer) (?x buyer) (sold(?good-id ?price ?x)))))
  defeasible-antecedent =
  ((buyer-settlement (inform (!x buyer) (?y buyer-accountant)
                             (payment(!price)))))
  consequent =
  ((obl !x (inform (!x buyer) (?y buyer-accountant)
                   (payment(!price))) buyer-settlement))
)

(define-norm sanction as
  antecedent =
  ((auction-room
    (commit (?x buyer) (?y auctioneer) (bid(?good-id ?price))))
   (< (credit !x) !price))
  defeasible =
  ((auction-room (inform (!y auctioneer) buyer (sanction(!x)))))
  consequent =
  ((obl !y (inform (!y auctioneer) buyer (sanction(!x)))))
)
```

Fig. 10. The `obligation2pay` and `sanction` norms

the illocutions $(s_{m+1}, \gamma_{m+1}), \ldots, (s_{m+n}, \gamma_{m+n})$ have *not* been uttered, the obligations $obl_1, \ldots, obl_p$ hold. Therefore, the rules have two components, the first one is the causing of the obligations to be activated (for instance winning an auction round by saying "mine" in a downwards bidding protocol, generating the obligation to pay), comprising parts 1 and 2 of the left hand side and the second is part 3 of the left hand side that defeats the obligations (for instance, paying the amount of money due for the round which was won).

Clearly, an external agent may not fulfill its obligations. As agents are autonomous and the institution accepts agents developed by other people, those agents cannot be forced to utter particular illocutions. Thus, it follows that the institutions cannot force agents to fulfill their obligations. However, the institution does know the obligations that each agent has acquired and can thus detect when an agent does not fulfill its obligations and hence violates the norms. Moreover, the institution can restrict the actions that an agent can carry out while it has not fulfilled some or all of its obligations.

As we can see in Figure 9 norms are specified in the following form: the actions that provoke the activation of the norm and the restrictions over illocution scheme variables expressed in the antecedent, the actions that agents must carry out in order to fulfill the obligations expressed in the defeasible antecedent, and the set of obligations expressed

An institution specification

dialogic-framework: a *name* of a dialogic framework
performative-structure: a *name* of a performative structure.
norms: a *list* of *name*s of norms.

Fig. 11. Elements of an institution specification

```
(define-institution
  fish-market as
  dialogic-framework = fm-dialogic-framework
  performative-structure = fm-performative-structure
  norms = (obligation2pay sanction)
)
```

Fig. 12. The FishMarket institution

on the consequent. The antecedent defines the set of illocutions that when uttered in the corresponding scene satisfying the boolean expressions will trigger the norm making the set of obligations expressed in the consequent hold. The defeasible antecent defines the illocutions that agent must utter in the defined scenes in order to fulfill the obligations.

Finally, we are in a position to combine a performative structure, a dialogic framework and a set of norms to construct an institution (see Figure 11 and 12).

4 Grounding the Language

Up to this point in the paper, we have presented an attempt at a formal description of an electronic institution and illustrated the use of that language with the relatively well-known FishMarket system. However, although we have described the static semantics of the language in informal terms, the specification is still abstract in computational terms. We see process algebra as a key intermediate level, which can formalize notions of place and action, sitting between our institution language and an actual executable form of the institution and which can establish a verifiable link between the different levels of abstraction. We also observe that the type systems and logics which abstract from the ambient calculus offer suitable frameworks in some cases for expressing low-level norms of limited scope, such as mobility, message types and message orderings (this view is expanded upon in [20]). Thus, we have four objectives in trying to use process algebra with locations to formalize the specification of institutions:

(i) To obtain a precise description of the components of the institution and their interactions.
(ii) To provide a formal framework within which the design can be verified.
(iii) To provide a formal framework within which the design can be validated against standard correctness requirements for distributed systems.
(iv) To provide a formal framework within which the design can be animated to verify institutional norms.

However, it is the first and last issues that hold the most interest for us: the first, because without it we can do nothing and the last because we see norms as the key to generating reputation for and trust in electronic trading institutions. We will now set out how we can relate the components of our institutional specification to the elements of process algebra with locations.

The two axes of the Ambient calculus (AC) are *communication* and *mobility*. The fundamental unit of AC is the notion of an ambient, which is a place within which processes may interact by writing messages into the ambient and reading them from it—hence reading and writing are decoupled and asynchronous. It is not possible for processes in different ambients to interact. Thus communication is a localized activity and it is only via movement that two processes in different ambients can arrive in the same ambient and hence interact. The unit of mobility is the ambient—not individual processes, but rather, a collection of processes and the messages that may be in transit between them. A process within an ambient may execute a capability to (make the whole ambient) enter a sibling ambient or move out of its enclosing (parent) ambient—these are called objective moves, because it is the ambient that moves itself—or an ambient may be dissolved, unleashing its constituents into the enclosing ambient—a subjective operation, because a process executes a capability in one ambient to carry out the operation on another. The effect of each of these operations is conveniently imagined as reorganizations of a tree (see Figure 15), where `in` detaches the sub-tree rooted at the moving ambient and re-attaches it as a child of the target ambient, `out` detaches as for enter and re-attaches it as a child of the parent of the parent and `open` attaches all the children of the subject ambient as children of the ambient performing the open. A sequence of mobility operations is called a path or a capability and may be passed as a message from one process to another. Attempting a move, when the target ambient is not present (i.e. as sibling or parent) causes the process executing that operation to block until the named ambient appears. However, other processes may continue to execute and the ambient itself may still undergo other objective or subjective moves.

From the above brief sketch of (untyped) AC, there is an attractive mapping of scenes to (immobile) ambients, where the conversation can take place via messages, and of agents to (mobile) ambients, which move from scene to scene given the appropriate capability. The situation improves further when we see what work has been started on type systems for AC, since these provide the basis for verifying some forms of norm. Exchange types [8], permit us to specify what may be read and written within an ambient, allowing us to specify which illocutions may be exchanged. An extension of this system [1], which as well as formalizing the idea of polymorphic exchange types, develops the notion of orderly communication being a sequence of types where the sequence relates to the passage of time. In our context, this may correspond to the progression of the conversation. Building on exchange types, Cardelli *et al.* [7], tackled the issue of describing the mobility of ambients, which for us means we can declare, for instance, that the various scenes are immobile (not necessarily always desirable: it may be preferable for an instance of, say, a bilateral negotiation scene to go to the two participants and then permit them to enter) and which ambients (scenes) the various agents may cross (enter/exit). However, while types may be adequate for some relatively simple security norms—who goes where, says what—they cannot capture the more complex norms, such as the examples given above for the FishMarket, which sit in logic and may have modal, temporal or deontic aspects to them. There has been an initial attempt to develop

$$\begin{array}{lll}
P ::= \mathbf{0} & \text{inactivity} \\
\quad \mid\ P \mid P & \text{composition} \\
\quad \mid\ !P & \text{replication} \\
\quad \mid\ (\nu\ x_1, \ldots, x_n)\ P & \text{restriction} \\
\quad \mid\ \alpha\ .\ P & \text{action} \\
\quad \mid\ n[P] & \text{ambient} \\
\quad \mid\ (x) & \text{bind input} \\
\quad \mid\ \langle\alpha\rangle & \text{output}
\end{array}
\qquad
\begin{array}{lll}
\alpha ::= x & \text{variable (read)} \\
\quad \mid\ n & \text{name (new)} \\
\quad \mid\ \mathit{in}\ \alpha & \text{enter } \alpha \\
\quad \mid\ \overline{\mathit{in}}\ \alpha & \text{allow enter } \alpha \\
\quad \mid\ \mathit{out}\ \alpha & \text{exit } \alpha \\
\quad \mid\ \overline{\mathit{out}}\ \alpha & \text{allow exit } \alpha \\
\quad \mid\ \mathit{open}\ \alpha & \text{open } \alpha \\
\quad \mid\ \overline{\mathit{open}}\ \alpha & \text{allow open } \alpha \\
\quad \mid\ \epsilon & \text{empty path} \\
\quad \mid\ \alpha\ .\ \alpha' & \text{path}
\end{array}$$

Fig. 13. Ambient calculus syntax from [8] with co-actions from [14]

$$\begin{array}{rcll}
n[\mathit{in}\ m\ .\ P \mid Q] \mid m[\overline{\mathit{in}}\ m\ .\ R] & \longrightarrow & m[n[P \mid Q] \mid R] & \text{(in)} \\
m[n[\mathit{out}\ m\ .\ P \mid Q] \mid \overline{\mathit{out}}\ m\ .\ R] & \longrightarrow & n[P \mid Q] \mid m[R] & \text{(out)} \\
\mathit{open}\ n\ .\ P \mid n[\overline{\mathit{out}}\ Q] & \longrightarrow & P \mid Q & \text{(open)} \\
(x)\ .\ P \mid \langle\alpha\rangle & \longrightarrow & P\{x, \alpha\} & \text{(communication)}
\end{array}$$

Fig. 14. Safe Ambient calculus reduction rules

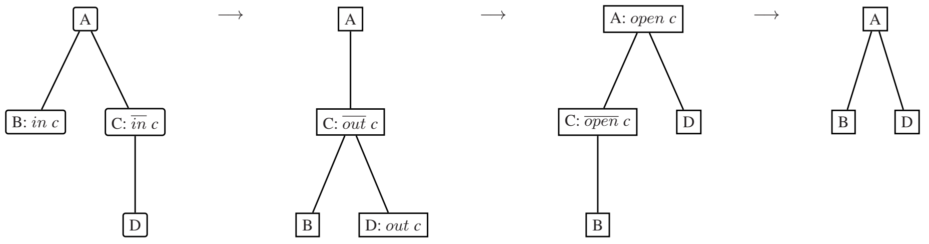


Fig. 15. Ambient calculus mobility operations

a modal logic for AC, which would permit properties such as "there is at most one staff agent in each scene", "every governor eventually reaches the exit scene" to be specified. Also of relevance is the temporal logic model checker developed for PoliS [10]. A fuller discussion of these issues, as well as a consideration of the Seal Calculus [9] appears in [21,20]. A summary of the syntax of AC appears in Figure 13, from which it will be observed that we have adopted the extension by co-actions, introduced in Safe Ambients [14]. Under this variant of AC, for every action (*in*, *out*, *open*) there is a corresponding co-action ($\overline{\mathit{in}}$, $\overline{\mathit{out}}$, $\overline{\mathit{open}}$) and for any action to succeed, the collaborator in the action (a sibling, child or parent ambient, respectively) must engage in the corresponding co-action, that is both parties to the ambient operation must synchronize.

4.1 Agent Communication

Our consideration of communications begins with the following two assumptions: that the external agent will be immobile (or leave an immobile representative/proxy) within the market for the duration of all its transactions and that the market representative—the governor—will move between the scenes of the market, interacting with the staff agents

$$\textsc{Market} \triangleq \textsc{Buyers-Admission} \mid \textsc{Sellers-Admission} \mid \textsc{Auction-Scene}$$
$$\mid \textsc{Buyers-Settlement} \mid \textsc{Buyers-Settlement}$$

Fig. 16. High-level market specification

and communicating with the external agent it represents. Thus a high level specification of the market is the composition of the different scenes (see Figure 16)

For each scene, a scene manager (SM) provides the following services:

1. mediation of local one-to-one and broadcast message traffic
2. management of agent transport
3. relay of messages between governors and associated external agents

A complete specification of such a SM is too long for the space available, so instead we restrict ourselves to examining the interface provided in each case and a sketch of the specification, looking at the infrastructure for scenes and their interaction with governors.

As noted above each ambient contains a single anonymous channel, communication is asynchronous and with the use of disjoint sums, that single channel becomes, in effect, multiple named channels, or rather a pool which uses type information to match messages with read requests (a similar but more detailed observation on polymorphism and ambients appears in [1]) and starts to look rather similar to the ideas outlined in SecureSpaces [4]. In spirit however, it is much closer to a classical AI component: the blackboard. Thus, our solution to communication takes that principle and implements a simple blackboard at scene ambient level, through which the sub-ambients communicate using *get* and *put* operations.

Thus, in order to send a message, a governor needs to eject an ambient which can post a message on the SM's blackboard, but since for the purposes of this experiment, we are not using objective moves, it is a matter of creating an ambient named *put* which will subsequently move out of the governor, and be dissolved by the SM, so unleashing the *inform* message for posting on the blackboard. Meanwhile, the SM reads the messages posted on the blackboard. In the case of an *inform* message, it checks to see if the intended recipient is a sub-ambient (i.e. another governor), and if so, constructs a *get* ambient containing the message, which enters the target ambient. The target ambient dissolves the *get*, unleashing the *inform* message, which gets posted on the governor's internal blackboard and is subsequently processed. If the target is not a sub-ambient, the message will somehow be routed to the recipient, assuming it is known within the institution. The *broadcast* message is simply broken into multiple *inform* messages and the *enter* and *exit* messages perform book-keeping for message routing. Thus, the governor engages in the following activities:

– $put[\langle \texttt{inform}(m, \textsc{To}, self)\rangle \;.\; out\ self]$ puts a message on to the blackboard for the agent $\textsc{To}$. The SM will forward the message by wrapping it in a *get* ambient.
– $open\ get$ is used by an agent to open ambients from the SM's blackboard in order subsequently to be able to receive `inform` and `exit` messages.
– $put[\langle \texttt{enter}(self)\rangle \;.\; out\ self]$ informs the SM of an agent's presence (since moves are subjective in AC, the SM must be informed explicitly of an agent's presence, otherwise it will not receive any messages, but see also the next section).

```
SCENEMANAGER $\triangleq$
repeat {
 open put
 | (inform($m, a, b$)) . if $b \in$ subambients
                          then get[⟨inform($m, a, b$)⟩ . in $b$]
                          else route message $m$ from $a$ to $b$
 | (enter($a$)) . add $a$ to routing tables
 | (exit($a$)) . delete $a$ from routing tables . get[⟨leave(out self)⟩ . in $a$]
 | (broadcast($m, a$)) . $\forall b$ : ($b \in$ subambients) & ($a \neq b$), ⟨inform($m, a, b$)⟩
}

GOVERNOR $\triangleq$
repeat {
 open get
 | (inform($m, a$)) . process inform
 | (leave(move)) . prepare to move . move
 | other agent services
}
```

$msgs = inform + enter + exit + leave + broadcast$

$put/get : messages^{\curvearrowright}\varnothing[^{\curvearrowright}\{governors, staff\}, {}^{\circ}\varnothing, msgs]$

$\textsc{Governor} : governors^{\curvearrowright}\varnothing[^{\curvearrowright}\{scenes\}, {}^{\circ}messages, inform + leave + \text{others}]$

$\textsc{Admission} : scene^{\curvearrowright}\varnothing[^{\curvearrowright}\varnothing, {}^{\circ}messages, msgs]$

Fig. 17. Ambient scene manager and governor

$\textsc{Transport}(route\ passenger) \triangleq go[\overline{in}\ go\ .\ route\ .\ \langle \texttt{enter}(passenger)\rangle\ .\ \overline{open}\ go]$

$\textsc{Transport} : transporters^{\curvearrowright}\varnothing[^{\curvearrowright}\{scenes\}, {}^{\circ}\varnothing, \varnothing]$

Fig. 18. Transport agents

– *put*[⟨exit(*self*)⟩ . *out self*] informs the SM that an agent would like to depart and the reply will include a capability to exit the scene.

Following the typing conventions presented in [7], where a type is effectively a quadruple, defining the group of ambients that can be crossed (meaning can be entered or exited) objectively (see earlier discussion of objective versus subjective), those that can be crossed subjectively, those that can be opened and the types of messages that can be exchanged, we can assert types (see Figure 17) for (i) the blackboard messages (*put* and *get*), to say they do not undergo objective moves, but do cross governors and staff, open nothing and exchange messages of type *msgs*, where we have additionally defined *msgs* as the sum of *inform*, *enter*, *exit*, *leave* and *broadcast* (ii) for the governor to say they do not undergo objective moves, but do cross scenes, open messages and exchange messages of type *msgs* and other (unspecified) internal types and (iii) for the scenes to say they do not undergo objective moves, do not cross anything (i.e. they are immobile), but do open messages and exchange messages of type *msgs*. Thus, some very basic norms can be verified by use of a type system.

4.2 Agent Mobility

In the scene manager specification, a governor is only able to move after it has sent an `exit` message to the scene manager, in reply to which it should receive a `leave` message, containing a capability to move out of the scene. Although this enforces some coordination between scene manager and governor, it is generally undesirable to issue capabilities freely, since they can be communicated and re-used. In consequence we have adopted a novel solution involving specialized transport ambients which are created on demand with fixed destinations, issue a capability for the agent that wants to move to enter it, and then dissolve the transport ambient upon arrival at its destination, after also generating a registration message for the new scene manager.

Synchronization is necessary at both ends of the journey, because the transporter should only start to move once the passenger ambient has entered it, furthermore, it should only be dissolved once it has arrived at the destination. This is elegantly resolved by using co-actions, and so we can specify synchronization on the arrival of the passenger ($\overline{in}$ *go*), then follow the path specified in the *route* capability, and finally synchronize on the dissolving of the ambient on arrival at the destination ($\overline{open}$ *go*)—see Figure 18.

5 Related Work

Recently there is a growing interest in incorporating organizational concepts into multi-agent systems, with the purpose of considering organization-centered designs of multi-agent systems. For instance, in [25] we find a methodology for agent-oriented design and analysis founded on the view of a system as a computational organization consisting of various interacting roles, although the notion of organization structure is only implicitly defined. Due to the complexity of the design and development of multi-agent systems some languages have been proposed for modelling agent based organizations [3] and [15]. The first presents a language covering the definition of organizations, roles, agents, obligations, goals, and conversations as well as a system for inferring and executing coordinated agent behaviours in multi-agent applications. Although supporting the definition of organizations, this approach exhibits a definite agent-centred view and, for example, negotiations among agents assume that an agent can let others know about its list of obligations and inter-dictions. Moss et al. [15] developed SDML (Strictly Declarative Modeling Language), a multi-agent object-oriented language for modelling organizations which is theory-neutral with respect to the capabilities of agents and, furthermore, includes a library for alternate agent architectures. One of its most distinctive features, from our point of view, is that within organizations there are predefined linkages among agents and predefined roles in which knowledge is embedded with the purpose of constraining behaviour.

In respect of process algebra, we believe there is no related work on its application to the kind of modelling we have presented here. However, in relation to the work cited above on type systems and static analysis for process algebra, the main omission is an assessment of [16] on the use of Flow Logic to discover or establish properties of process algebraic specifications from static analysis. We hope to remedy this in the near future. In terms of a practical realization of process algebraic models, there are three candidates, namely Ambients on top of JOCAML [12], the JavaSeal kernel [5] and Nomadic Pict

[24], all of which are relatively experimental in nature. Tools for type systems and logics for process algebra are also few, largely due to the diversity of approaches currently under exploration, although the temporal logic system of [10] seems promising.

Acknowledgements

Julian Padget was partially supported by a European Study Visit award from the Royal Society and the Consejo Superior de Investigaciones Cientificas (CSIC). Marc Esteva enjoys the CIRIT doctoral scholarship 1999FI-00012. The research reported in this paper is supported by the the SLIE project IST-1999-10948 and the Spanish CICYT project eINSTITUTOR (TIC2000-1414).

References

1. Torben Amtoft, Assaf Kfoury, and Santiago Pericas-Geertsen. What are polymorphically-typed ambients? In David Sands, editor, *Programming Languages and Systems, 10th European Symposium on Programming, ESOP 2001, volume 2028 of Lecture Notes in Computer Science*, pages 206–220. Springer Verlag, 2001.
2. J. L. Austin. *How to Do Things With Words*. Oxford University Press, 1962.
3. Mihai Barbuceanu, Tom Gray, and Serge Mankovski. Coordinating with obligations. In *Proceedings of the Third International Conference on Autonomous Agents (AGENTS'99)*, pages 62–69, 1998.
4. C. Bryce, M. Oriol, and J. Vitek. A Coordination Model for Agents Based on Secure Spaces. In P. Ciancarini and A. Wolf, editors, *Proc. 3rd Int. Conf. on Coordination Models and Languages, volume 1594 of Lecture Notes in Computer Science*, pages 4–20, Amsterdam, Netherlands, April 1999. Springer-Verlag, Berlin. revised into Coordinating Processes with Secure Spaces and to appear in Science of Computer Programming (Autumn 2001).
5. Ciaran Bryce and Jan Vitek. The JavaSeal mobile agent kernel. In *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*, Palm Springs, CA, USA, October 1999.
6. L. Cardelli. Mobile Ambient Synchronization. Technical Report SRC Tech Note 1997-013, Digital, July 1997.
7. Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Ambient groups and mobility types. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, Proceedings of the International IFIP Conference TCS 2000 (Sendai, Japan)*, volume 1872 of LNCS, pages 333–347. IFIP, Springer, August 2000.
8. Luca Cardelli and Andrew D. Gordon. Types for mobile ambients. In ACM, editor, POPL '99. *Proceedings of the 26th ACM SIGPLAN-SIGACT on Principles of programming languages, January 20–22, 1999, San Antonio, TX*, ACM SIGPLAN Notices, pages 79–92, New York, NY, USA, 1999. ACM Press.
9. G. Castagna and J. Vitek. Seal: A framework for secure mobile computations. In H. Bal, B. Belkhouche, and L. Cardelli, editors, *Internet Programming Languages*, number 1686 in LNCS, pages 47–77. Springer, 1999.
10. P. Ciancarini, F. Franzè, and C. Mascolo. Using a Coordination Language to Specify and Analyze Systems Containing Mobile Components. *ACM Transactions on Software Engineering and Methodology*, 9(2):167–198, 2000.
11. M. Esteva and C. Sierra. Islander1.0 language definition. Technical report, IIIA-CSIC, 2001.

12. Cédric Fournet, Jean-Jacques Lévy, and Alain Schmitt. An asynchronous distributed implementation fo mobile ambients. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, Proceedings of the International IFIP Conference TCS 2000 (Sendai, Japan)*, volume 1872 of LNCS, pages 348–364. IFIP, Springer, August 2000.
13. Michael R. Genesereth and Richard E. Fikes. Knowldege interchange format version 3.0 reference manual. Technical Report Report Logic–92–1, Logic Group, Computer Science Department, Standford University, June 1992.
14. Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. In *Conference Record of POPL'00: The 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 352–364, Boston, Massachusetts, January 19–21, 2000.
15. S. Moss and B. Edmonds. A formal preference-state model with qualitative market judgements. *Omega – the International Journal of Management Science*, 25(2):155–169, 1997.
16. Flemming Nielson and Hanne Riis Nielson. Shape analysis for mobile ambients. In *Conference Record of POPL'00: The 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts, Jan 2000. ACM, ACM Press.
17. P. Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona, 1997.
18. Pablo Noriega and Carles Sierra. Towards layered dialogical agents. In *Third International Workshop on Agent Theories, Architectures, and Languages, ATAL-96*, 1996.
19. Douglass C. North. *Institutions, Institutional Change and Economic Performance*. Cambridge University Press, 1991.
20. Julian Padget. Modelling simple market structures in process algebras with locations. *Artificial Intelligence and Simulation of Behaviour Journal*, 1(1):87–108, 2001. to appear.
21. Julian Padget. Modelling simple market structures in process algebras with locations. In Luc Moreau, editor, *AISB'01 Symposium on Software Mobility and Adaptive Behaviour*, pages 1–9. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour, AISB, 2001. ISBN 1 902956 22 1.
22. J-A. Rodríguez. *On the Design and Construction of Agent-mediated Institutions*. PhD thesis, Universitat Autonoma de Barcelona, July 2001.
23. J. R. Searle. *Speech acts*. Cambridge U.P., 1969.
24. Pawel Wojciechowski and Peter Sewell. Nomadic pict: Language and infrastructure design for mobile agents. In *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*, Palm Springs, CA, USA, October 1999.
25. Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (AGENTS'99)*, May 1999.
26. G. H. Wright. Deontic logic. Mind, 60:1–15, 1951.

Simple Negotiating Agents in Complex Games: Emergent Equilibria and Dominance of Strategies

Peyman Faratin[1], Mark Klein[1], Hiroki Sayama[2], and Yaneer Bar-Yam[2]

[1] Sloan School of Management, Center for Coordination Science,
MIT, Cambridge, MA, 02139, USA,
{peyman,m_klein}@mit.edu
[2] New England Complex Systems Institute,
24 Mt. Auburn St., Cambridge, MA 02138, USA,
{sayama,yaneer}@necsi.org

Abstract. We present a simple model of distributed multi-agent multi-issued contract negotiation for open systems where interactions are competitive and information is private and not shared. We then investigate via simulations two different approximate optimization strategies and quantify the contribution and costs of each towards the quality of the solutions reached. To evaluate the role of knowledge the obtained results are compared to more cooperative strategies where agents share more information. Interesting social dilemmas emerge that suggest the design of incentive mechanisms.

1 Introduction

Arguably one of the most significant contributions of AI to the theory of computation has been models of the relationship between knowledge, computation and the quality of solution (henceforth referred to as the K-C-Q relationship) [24,18]. In this article computation is equivalent to the process of searching a space of possibilities [21]. AI has shown that there exists a hierarchy of tradeoffs between K, C and Q, with models that achieve perfect optimal results but at the cost of requiring omniscience and unbounded agents, to models that sacrifice optimality of Q for a more realistic set of requirements over K and C [22]. Different agent architectures are then entailed from different K-C-Q relationship theories. However, the rise of Distributed Artificial Intelligence (DAI), and Multi-Agent Systems (MAS) in particular, has given rise to the need for new models of this relationship, of not only the *local* K-C-Q relationship for the agent itself but also the *social* K-C-Q relationship. In social systems, the underlying relationship is regulated through the protocol of interaction.

Game theoretic models have emerged as the most popular candidate social models in MAS [15,20,23]. However, in such prescriptive models desirable solutions are achieved at the cost of make very restrictive assumptions over both K and C [4,10,6]. In the types of problems that interests us K and C are limited because not only do agents have an incomplete knowledge of their own utility functions but they also have, at best, imperfect and at worst no knowledge of the others' utility functions. We say that an agent's knowledge of its own utility function is incomplete when it can not compare the utility of a point in the utility landscape with all the other points and searching this landscape exhaustively is computationally prohibitive. Additionally, agents rarely have full knowledge of others' utility landscape. The best an agent can do is to reason with imperfect knowledge by forming approximations of others' utilities (and later correcting for imperfections using some update rule such as Bayesian rule [19]). In the worst case, agents have no knowledge of the others utility landscape. In either case, lack of perfect knowledge results in a process of negotiation that requires agents to search for mutually acceptable solutions. Therefore the problem is how to design search algorithms that are not computationally prohibitive but also result in good solutions given limited knowledge.

Descriptive (or *evolutive*) models of evolutionary game theory are emerging as alternatives that make fewer restrictive assumptions as well as providing models of the dynamics of interactions [25,1,4]. In these models the individual *is* merely a strategy which is subjected to survival criteria in a population of other strategies. The problems associated with the prescriptive models are eliminated by replacing the agents with simple stimulus-response machines. Under this methodology coordination *emerges* from interactions between simple agents. In this paper we adopt a similar methodology as the evolutive approach to address the computation and knowledge complexities involved in the problem of large scale and interdependent multi-issued contract negotiation between autonomous agents in an open system. In particular, we show that distributed decision making using two simple approximation methods (the hillclimbing and the simulated annealing) can result in interesting social and local dynamics when the knowledge, computation and the quality of the solution is regulated by a voting protocol.

Section 2 is a brief introduction and overview of the contract negotiation problem and its features. We then describe a mediated negotiation protocol in section 3 followed by a model of the problem objective function in section 4. Sections 5 and 6 then presents a pair of simple selfish (local and hence knowledge poor) and cooperative (global and hence knowledge richer) evaluation strategies respectively. In the penultimate section 7 we presents the simulation results. Finally, section 8 identifies the weaknesses and the future directions of research.

2 The Contracting Problem

The particular complexity of interest for us is the knowledge and computation involved in open system multi-party contract negotiation over multiple clauses (we refer to a contract clause as an issue). In particular, we are interested in design of distributed agent decision algorithms that select a negotiated outcome that are both individually and socially good given that the agents are bounded in both knowledge and computation. In multi-issue negotiation computation is often bounded because of *scale* and *issue dependencies*. Scale is a problem because even a relatively simple contract can easily involve 100 issues. Even with only two alternatives for each issue, the size of the search space is roughly 10^{30} possible contracts, too large to be explored exhaustively. This implies that agents will not know which contracts are best for them a priori. Therefore they have to search the contract design space. The problem of issue dependency occurs for the following reason. If the issues being negotiated over are assumed to be independent (i.e., the utility to an agent of a given issue choice is independent of what selections are made for other issues), then the utility function that aggregates the individual utilities is a linear one (often additive). This assumption significantly simplifies the agents' local decision problem of what issue values to propose in order to increase their own utility: they simply need to "hillclimb" in a straight line up the utility slope. Almost all work to date on computational negotiation algorithms has in fact focused on the independent issue case [11,9]. Real world contracts, however, are highly inter-dependent. For example, in a supply chain the output of one sub-step in a process will often have to match the input of the next sub-step, so the utility of one choice is highly dependent on the other choice. When issue interdependencies exist, the utility function for the agents exhibits multiple local optima, and the process of finding a satisfactory contract becomes much more complicated because it will often make sense during the search process to consider contracts with low utility while moving towards contracts with potentially much higher utilities.

Additionally, in open systems knowledge is scarce because of information exchange constraints. MAS negotiation approaches do not assume that agents will be cooperative and make choices based on their impact on global utility, but rather that they will be purely self-interested [8]. This makes negotiation well-suited to open systems where the participants can be diverse and we can not assume that all of them will be cooperative or even benevolent [10]. Because of the competitive nature of negotiation, however, agents will typically wish to reveal as little information as possible about themselves to other agents. Therefore, computation is knowledge poor.

Another consequence of the open system assumption underlying negotiation approaches is that, since we can not assume that agents will be altruistic, we must design negotiation protocols such that the individually most beneficial negotiation strategies also produce the greatest social welfare (i.e., the greatest aggregate utility summed over all agents) [20,23]. In other words, we want the socially most beneficial strategy to also be the individually dominant one so that most agents will tend to use it. There has been significant success achieving these property in the independent issue case, but the specification of dominant strategies becomes more complicated, as we shall see, for the interdependent issue case.

The challenge therefore is to define contract negotiation algorithms that are suited to large design spaces with interdependent issues but are sensitive to the information exchange constraints and strategy dominance concerns important in open system contexts. Approximation techniques to such hard problems have been successful in the past for other such hard problems such as machine vision [3]. We present agent strategies as an approximate optimization algorithms in sections 5 and 6 after an informal description of the protocol of negotiation below.

3 The Negotiation Protocol

A contract S, is an N dimensional boolean vector where $S_i \in \{-1, +1\}$, represents the presence or absence of a "contract clause" i. In the simulations the size of the vector S was set to 100 bits, corresponding to a space of 2^{100}, or roughly 10^{30} possible contracts.

The contract search policy is encoded in the negotiation protocol. Because generating contract proposals locally is both knowledge and computationally expensive (see [11] for an approach to this problem) we adopt an indirect interaction protocol between two agents by delegating the contract generation process to a centralized mediator [12]. The mediator proposes a contract S^t at time t. Each agent then votes to accept or reject S^t. If both vote to accept, the mediator iteratively mutates the contract S^t and generates S^{t+1}. If one or both agents vote to reject, a mutation of the most recent mutually acceptable contract is proposed instead. The process is continued until the utility values for both agents become stable (i.e. until none of the newly contract proposals offer any improvement in utility values for either agent). In this manner, search of the joint and local utility landscapes of the agents is iteratively carried out by the mediator who incrementally modifies *a* randomly selected "contract clause" from a *present* state $(+1)$ to a *absent* state (-1) or vice versa. Agents then decide whether to accept or reject this new single "step" in their local utility landscape. Note that this approach can straightforwardly be extended to a N party (i.e. multi-lateral) negotiation.

4 The Objective Functions

At the end of each modification agents must vote to either accept or reject a contract. This decision is based on the utility of a contract. The utility is defined as the linear combination of all the pairwise influences between issues. That is:

$$u = \sum_{i,j} J_{ij} S_i S_j \qquad (J_{ij} = J_{ji}, J_{ii} = 0) \tag{1}$$

where u is the local contract utility of an agent, and $J_{ij} \in [-1, +1]$ is the utility influence matrix between issues i and j, which represents the utility increment or decrement caused by the presence of a given pair of issues. We assume that this pairwise effect is symmetric and not defined for $i = j$, as shown in parentheses. Each negotiation agent has a local and private J_{ij} matrix which is different to that of all other agents. We do not include the increment or decrement by the presence of a particular clause itself, since they are linear terms that do not affect the non-linearity of the utility space that is of our main interest. Also note that interactions among three or more clauses are omitted for simplicity, since the search is already complex enough for problems involving hundreds of pairwise relations. The key issue we want to model is the conflicts between agents, and they are guaranteed in this simple model when the objective function is defined with different distributions of J_{ij} for different agents.

Given the utility of a contract agents must then decide in a distributed manner whether to accept or reject a contract S^t at time t. To examine the role of knowledge (or willingness to share information) on the quality of solution in this model of negotiation we distinguish two types of evaluation strategies, local and global. These strategies, described in more detail in sections 5 and 6 respectively, have counterparts in game theory [13].

5 Local Agent Strategies

In a local evaluation strategy agents do not share any information and evaluate the generated contract S in a distributed or local fashion. Thus, what is being optimized is the *individual* utility of the agent u given in equation 1. Specifically, agents' decision to accept or reject a contract S^t is a function of the magnitude of the difference between S^t and S^{t-1} defined as $\Delta u^k(S^t, S^{t-1}) = u^k(S^t) - u^k(S^{t-1})$, for agent k at time t.

Two computationally inexpensive decision algorithms were evaluated: a hillclimber and a simulated annealer. A hillclimber decision rule is defined as:

$$p^k(S^t) = \begin{cases} 1 & \text{if } \Delta u^k(S^t, S^{t-1}) > 0.0 \\ 0 & otherwise \end{cases} \tag{2}$$

where $p^k(S^t)$ is the probability that the agent k will accept the given contract S^t generated from the previous contract S^{t-1}. Therefore, a hillclimbing search evaluation function only accepts a contract if and only if the utility of the contract S increases monotonically when *an* issue is changed.

However, the formulation of the objective function given by equation 1 has multiple optima. Therefore, this steepest ascend algorithm is known to be incapable of escaping local maxima of the utility function. Search success can be improved by adding thermal noise to this decision rule [14]. The concrete probabilistic rule used to simulate thermal noise is:

$$p^k(S^t) = \begin{cases} 1 & \text{if } \Delta u^k(S^t, S^{t-1}) > 0.0 \\ e^{\Delta u^k(S^t, S^{t-1})/T} & otherwise \end{cases} \tag{3}$$

where $p^k(S^t)$ is the probability that the agent k will accept the given contract S^t at temperature T. The policy of decreasing T with time is called simulated annealing [14]. The success of the rule is based on its elimination of barriers between local minima by giving a small chance to large utility configurations. This decision rule is known to reach utility equilibrium states when each issue is changed with a finite probability and time delays are negligible [14].

Like evolutive models these decision rules completely specify the agents. We categorize these rules in to selfish strategies (equation 2) accepting S^t iff $u(S^t) > u(S^{t-1})$, or cooperative (equation 3) by accepting a contract S^t when $u(S^t) \leq u(S^{t-1})$ with a certain probability determined by the thermal noise. Note, that if we define accept and reject as the only *pure* strategies, then both the hillclimbing and simulated annealing profiles are equivalent to implementing a *mixed* strategy of accept *and* reject, one deterministically (hillclimber) and the other according to a probability determined by the thermal noise [13]. Furthermore, although not explicitly modeled the thermal noise variable can be interpreted as encoding in to the behaviour of agent its willingness to take risks in return for future possible gains. The higher T the more risk loving and, conversely, the lower T the more risk averse the agent.

6 Coalition Strategies

In the above section agents' evaluation of S^t is distributed and local. Global evaluation strategies were also considered to assess the role of knowledge on the quality of computation and compare the results obtained from distributed v.s centralized decision making (the latter approach is the most popular with OR algorithms). We call this global evaluation a coalition strategy because agents have *agreed* prior to the game to a strategy of sharing their local utility for a contract and optimizing some social welfare function (simply defined as the sum of the local utilities) during the game.[1] This formulation is equivalent to agreements made before a game on a pair of strategies that maximizes the joint social welfare for two player games in cooperative game theory [16]. Local strategies are equivalent to non-cooperative games where there is no pre-negotiation agreements before the ensuing game [17]. Note that whereas there are a total of 2^N possible pairs of combination of hillclimbing simulated annealing strategies for N agents, in cooperative evaluations there are only two strategies available, hillclimbing or simulated annealing. As will be shown quantitatively later a coalition removes the gaming part of the interaction.

7 Simulations: Procedures and Results

Simulations were run with two agents referred to as a and b. The contract length N was set to 100 where each bit was initialized to a value $\{-1, +1\}$ randomly with a uniform distribution. The utility influence matrices J_{ij}^k for agent k was also randomly selected to be in the interval $[-1, +1]$ with a uniform distribution. Different influences matrices were used for each simulation run, in order to ensure our results were not idiosyncratic to a particular run. The initial temperature was set to 10 and decreased in steps of 0.1 to 0. Final average utilities were collected for 100 runs for each temperature decrement. Since there were only two agents the simulation pairing of local strategies making up the encounters in negotiation were: 1) a pair of local hillclimbers, 2) a pair of local simulated annealers and 3) one local simulated annealer and one local hillclimber (the asymmetric case).

Figure 1 shows the results of the social welfare for different pairings of both local and coalition strategies. Relatively best results were observed when agents were cooperative and agreed on a joint annealing strategy to optimize the sum of their local utilities (filled circle) rather than optimize only their local utilities (all unfilled data points). However the final benefits of a coalition was lower when the optimization strategy was hillclimbing (filled triangles up) than a pair of local annealing strategies (unfilled circles), even though initially hillclimbing cooperatively over the joint utilities leads to better results quicker. These observations, and in general comparisons of the annealers and hillclimbers independently of the level of cooperation, suggests the natural C-Q tradeoff between search time and solution quality, where initially searching more (annealers at high temperatures), although costly, will in the long term result in better solution quality. This is confirmed by the observation that the presence of a local annealer agents always increases the final social welfare. The social welfare for the two local annealer case (unfilled circle in figure 1) was roughly 40% greater than that of the two local hill-climber case (unfilled triangle up in figure 1), and the asymmetric case (unfilled diamond in figure 1) produced a smaller but still significant 15% improvement over the local hill-climbers. The results also show that both local and coalition hill-climbers

[1] Note, this social welfare formulation is the pareto-optimal concept of efficiency [7]. However, unlike Nash bargaining solution [16] pareto-optimality, although efficient, does not imply fairness. Although we have some results on this issue we do not address the problem of fairness in the coalition in this paper.

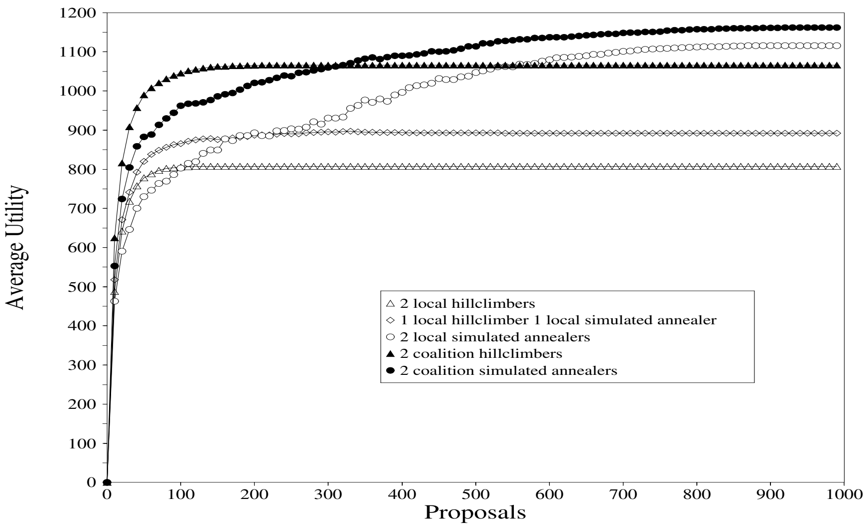


Fig. 1. Social Payoffs for both local and coalition strategies

reach equilibrium sooner than the annealers. Local hill climbers for example typically reached stability after roughly 100 proposal exchanges, while the annealers approached equilibrium values after roughly 800 proposal exchanges. This is to be expected since the combination of hillclimbing search to either the global or local welfare is thresholded because hill climbers simply climb to the top of the *closest* utility optimum from the starting position and then stop, while annealers can, when at a high temperature at least, explore different multiple optima in the utility function. One might argue that the gain in utility achieved by using annealers was relatively unimportant given the time taken to achieve these results. However, the complexity of computation used to produce the observed quality of results is insignificant given the simplicity of the simulated annealers. This is demonstrated by noting the run times were on the order of seconds. Furthermore, although coalition hillclimbers were dominated by local annealers on the long term, the dominance of both coalition annealing and hillclimbing strategies over local strategies, shows that sharing information is generally better than not, thereby quantitatively confirming the K-Q relationship that more knowledge results in better solution quality. This relationship can be seen by comparing the social welfare obtained when agents make decisions in a coalition (filled data points) with decisions made locally (unfilled data points).

Individual payoffs were then examined to investigate the relationship of C-Q with local utility metric of Q. Figure 2 shows the observed individual payoffs for different pairs of local strategies. One observation is that if the other agent is a local hill-climber, an agent is then individually better off being a local hill-climber, but fares very badly as local annealer. If the other agent is an annealer, the agent fares well as an annealer but does even better as a hillclimber. The highest social welfare, however, is achieved when both agents are annealers. This pattern can be readily understood as follows. At high virtual

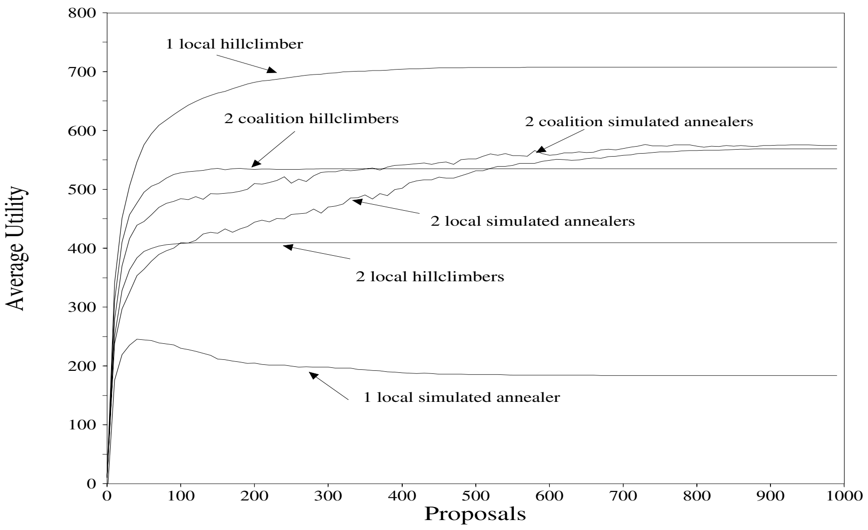


Fig. 2. Individual Payoffs for local and coalition strategies

temperature an annealer in the asymmetric case accepts almost all proposed contracts independently of the cost-benefit margins. The asymmetry means that the cooperation of annealers *permits* more exploration, and hence arrival to higher optima, of hillclimber's utility landscape. However, this cooperation is not reciprocated by hillclimbers who act selfishly. Therefore, gains of hillclimbers are achieved at the cost of the annealer. Finally, the local benefits of coalition strategies closely reflect the same rank orderings as the social benefit results shown in figure 1. Figure 3 represents the underlying game as a matrix of final observed utilities for all the pairings of hillclimber and annealer strategies. The results confirm that this game is an instance of the prisoner's dilemma game [2], where for each agent the dominant strategy is hillclimbing. Therefore, the unique dominating strategy is for both agents to hillclimb. However, this unique dominating strategy is pareto-optimally dominated when both are annealers. In other words, the single Nash equilibria of this game (two hillclimbers) is the only solution not in the Pareto set.[2]

	Agent b Annealer	*Agent b Hillclimber*
Agent a Annealer	550/550	180/700
Agent a Hillclimber	700/180	400/400

Fig. 3. Payoff Matrix of the Game

[2] Note, for von Neumann and Morgernstern utility functions, the same underlying game structure is achieved through any affine transformations of the utility values.

One possible solution to this problem is to anneal or hillclimb not over the individual utilities (the distributed case) but instead over the joint utilities (the coalition case, figure 2). Then an institution or some centralized trusted third party eliminates gaming between the agents altogether by centrally executing a commonly known strategy, and searching through the set of possible contracts *given* agents local utility functions. However, truthful revelation of utility functions is a strong assumption and problematic and requires the design of additional mechanisms that implement the Revelation Principle where agents are incented to truthfully reveal their true utility functions [5]. Alternatively, agents may be permitted to evolve towards the socially dominant strategies of the underlying game incrementally [25]. In the context here this means that the design process of negotiating machines will eventually lead to design of machines that implement the simulated annealing strategy.

8 Conclusions and Future Work

The main contribution of this work has been to demonstrate how simple descriptive models of interactions can account for the dynamics of negotiation and lead to the emergence of interesting social outcomes. Two simple and strategic agent evaluation algorithms (hillclimbing and simulated annealing) were presented, regulated by a mediated voting protocol, for solving complex and large scaled problem of multi and interdependent issued contract negotiation. We have shown that there exists a tradeoff between the quality of solution reached and the computations involved with hillclimbing and simulated annealing algorithms. More search led to better contracts when annealing over either the individual or the joint utility landscape. Conversely, less search leads to relatively poorer contracts when hillclimbing over either the individual or the joint utility landscape but at a relatively lower costs. We have also shown the benefits gained on the quality of solution in cooperative negotiations where more knowledge is shared. Furthermore, we have demonstrated how local decision making can lead to the prisoner's dilemma games, a problem that is relevant to any distributed synthesis task involving negotiation (not just contract formation).

There are a number of future directions. Work is in progress to benchmark the computational complexity of the developed algorithms against centralised algorithms from Operation Research and evaluating their optimality against solution concepts such as the Nash bargaining Solution and Pareto-Optimality. Additionally, work is under way to develop a better causal model of the parameters (such as the starting temperature, the number of issues, the cooling regime, etc.) that regulate the K-C-Q relationship which can then support decision making over parameter values. Other, more intelligent, combinatorial optimization techniques are also being evaluated that better implement a contract generation policy. Finally, in the longer term we aim to develop and comparatively analyze additional agent strategies such as reciprocation, learning and evolutionary rules that execute in distributed or centralized repeated interaction protocols.

References

1. R. Axelrod. *The Evolution of Cooperation*. Basic Books, Inc., Publishers, New York, USA., 1984.
2. R. Axelrod. *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton Press, Princeton, New Jersey., 1997.
3. D.H Ballard and C.M. Brown. *Computer Vision*. Prentice Hall, Englewood Cliffs, NJ, 1982.
4. K. Binmore. *Essays on the foundations of game theory*. Basil Blackwell., Oxford, UK, 1990.
5. K. Binmore. *Fun and Games: A Text on Game Theory*. D.C. Heath and Company., Lexington, Massachusetts, 1992.
6. C. Castlefranchi and R. Conte. Limits of strategic rationality for agents and MA Systems. In M. Boman and W. Van de Velde, editors, *Multi-Agent Rationality: Proceedings of the 8th European Workshop on Modeling Autonomous Agents in Multi-Agent World, MAAMAW'97*, number 1237 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1997.
7. G. Debreu. *Theory of Value: An Axiomatic Analysis of Economic Equilibrium*. Wiley, New York, 1959.
8. Edmund H. Durfee and Jeffrey S. Rosenschein. Distributive problem solving and multi-agent systems: Comparisons and examples. In *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence, Seatle, WA*, pages 94–104, Seatle, WA, July 1994.
9. H. Ehtamo, E. Ketteunen, and R. Hamalainen. Searching for joint gains in multi-party negotiations. *European Journal of Operational Research*, 1(30):54–69, 2001.
10. P. Faratin. *Auotmated Service Negotiation between Autonomous Computational Agents*. PhD thesis, Department of Electronic Engineering, Queen Mary andWestfield College, University of London, 2000.
11. P. Faratin, C. Sierra, and N. R. Jennings. Using similarity criteria to make negotiation trade-offs. In *Proceedings of the International Conference on Multiagent Systems (ICMAS-2000), Boston, MA.*, pages 119–126, 2000.
12. K. Fisher and W. Ury. *Getting to Yes: Negotiating Agreement without giving in*. Houghton Mifflin, Boston, MA, 1981.
13. R. Gibbons. *A Primer in Game Theory*. Harvester Wheatsheaf, New York, 1992.
14. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecci. Optimization by simulated annealing. *Science*, pages 671–680, 1983.
15. S. Kraus and D. Lehmann. Designing and building negotiation automated agent. *Computational Intelligence 11*, 11(1):132–171, 1995.
16. J. Nash. The bargaining problem. *Econometrica*, 18:155–162, 1950.
17. J. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–95, 1951.
18. J. Pearl. *Heuristics*. Addison-Wesley, Reading, MA, 1984.
19. H. Raiffa. *Decision Analysis: Introductory Lectures on Choices under Uncertainty*. Addison-Wesley, Reading, MA, 1968.
20. J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. The MIT Press, Cambridge, USA, 1994.
21. S. Russell and P. Norvig. *Artificial Intelligence: A modern approach*. Prentice Hall, Upper Saddle River, New Jersey, 1995.
22. S. Russell and E. Wefald. *Do the Right Thing*. The MIT Press, 1991.
23. T.W. Sandholm. Distributed rational decision making. In G. Weiss, editor, *Multiagent Systems*, pages 201–259. The MIT Press, Cambridge, Massachusetts, 1999.
24. H. A. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, Massachusetts., 1996.
25. J. Maynard Smith. *Evolution and the Theory of Games*. Cambridge University Press, Cambridge, 1982.

Optimal Negotiation Strategies for Agents with Incomplete Information

S. Shaheen Fatima[1], Michael Wooldridge[1], and Nicholas R. Jennings[2]

[1] Department of Computer Science,
University of Liverpool, Liverpool L69 7ZF, U.K.
{S.S.Fatima, M.J.Wooldridge}@csc.liv.ac.uk
[2] Department of Electronics and Computer Science,
University of Southampton, Southampton SO17 1BJ, U.K.
nrj@ecs.soton.ac.uk

Abstract. This paper analyzes the process of automated negotiation between two competitive agents that have firm deadlines and incomplete information about their opponent. Generally speaking, the outcome of a negotiation depends on many parameters—including the agents' preferences, their reservation limits, their attitude toward time and the strategies they use. Although in most realistic situations it is not possible for agents to have complete information about each of these parameters for its opponent, it is not uncommon for agents to have partial information about some of them. Under such uncertainty, our aim is to determine how an agent can exploit its available information to select an *optimal strategy*. Here, in particular, the optimal strategies are determined considering all possible ways in which *time* can effect negotiation. Moreover, we list the conditions for convergence when both agents use their respective optimal strategies and study the effect of time on negotiation outcome.

1 Introduction

Automated negotiation is a key form of interaction in systems composed of autonomous agents [3]. Given its ubiquity, such negotiations exist in many different shapes and forms (see [7] for a taxonomy). Here, however, we consider a particular class of automated negotiation; namely, competitive bargaining over a single issue (price) between two agents that both have firm deadlines. This is exemplified by the e-commerce scenario in which a buyer agent and a seller agent negotiate over the price of a good or service. The buyer clearly prefers a low price, while the seller prefers a high one (hence the competitive nature of the encounter). In addition to attempting to obtain the best price, agents also usually need to ensure that negotiation ends before a certain deadline. However, the end point may not be the only way in which time influences negotiation behaviour. Consider the case in which the service is provided immediately after negotiation ends successfully (say at price P and time T). In some situations, it is not sufficient merely for an agent to ensure that T is any time less than its deadline. This may be the case, for instance, because one of the agents, say the buyer, could be losing utility with time as a result of not getting the service. On the other hand, the seller may perhaps gain more utility by providing the service as late as possible. Thus, in this case, the seller tries to maximize

T (within the limit of its deadline) and the buyer tries to minimize T. In short, it is clear that agents can have different attitudes toward time.

Generally speaking, the most common time effects in bargaining situations are [6]:

- Discounting: Benefits received immediately by an agent are preferred to the same benefits received in the future.
- Bargaining Cost: The bargaining process itself may incur some cost to an agent.
- Sudden Termination: An agent may have a deadline beyond which it cannot continue negotiation.

In addition to time, the outcome of a negotiation typically depends on many other parameters; such as the agents' preferences, their reservation limits, and the strategies they use. Although in most realistic cases it is not possible for agents to have complete information about all of these parameters for its opponent, it is not uncommon to have partial information about some of them. For instance, an agent may have information about its opponent's preferences, or its deadline. In this paper, we focus on situations where an agent has the following information about its opponent [1]:

- A set of possible values for the opponent's reservation limit and a binary probability distribution over these values.
- A set of possible values for the opponent's deadline and a binary probability distribution over these values.

With this information an agent can optimize its utility from price and time. However we do not assume that agents have full information about the preferences of their opponent or the strategy that they use. It is known (common knowledge) that both agents use a strategy that varies their negotiation stance with time, but the particular type of *time dependent strategy* that an agent uses is not known to its opponent. Under such uncertainty, our aim is to determine how an agent can exploit the available information to select a strategy that maximises its expected utility. Moreover, when both agents have this information about one another, we determine the impact of this information on the outcome of negotiation. This analysis is important because it enables us to construct software agents that will *optimally* negotiate on behalf of users given their state of knowledge in a given context.

The remainder of the paper is structured in the following manner. Section 2 discusses related work. Section 3 describes the basics of our negotiation model. Section 4 determines the optimal strategies for agents with incomplete information about each other. In section 5 we analyze the outcome of negotiation when both agents use their respective optimal strategies. Finally, in section 6 we present the conclusions and outline some avenues for future work.

2 Related Work

Game theoretic research typically deals with coordination and negotiation issues by assuming that agents have complete information about each other and then giving pre-

[1] This information is *private* in the sense that the values that an agent has about its opponent are not known to the opponent.

computed solutions to specific problems [10]. However this perfect information assumption is limiting because uncertainty is endemic in most realistic applications. Harsanyi et al [2] give a generalized solution for two person bargaining games with incomplete information. However there is no notion of timing issues in their model. Another important model of strategic bargaining is the infinite horizon alternating offer game [11]. Since this has a unique solution, where agents agree on a split immediately, it has been applied to automated negotiation [5]. However while this model takes time into consideration, it again assumes perfect information. Faratin et al.'s negotiation framework [1] models time as agents' deadlines and is not based on the assumption that agents have perfect information. However in this model the agent's utility functions depend only on negotiation issues like price and quality, but are independent of time.

Perhaps the work that is most closely related to ours is that of Sandholm and Vulkan [12]. In their work on bargaining with deadlines, they consider the probability distribution over agent deadlines to be common knowledge. Specifically, they address the problem of splitting the price-surplus which is known to both agents. They show that the optimal strategy is one in which agents wait until the first deadline, at which point one agent concedes everything to the other. Thus agents only ever make two offers, they either demand the entire surplus or no surplus at all. This gives the entire surplus of price to the agent with the longer deadline. But because of the offers made by agents, the deadline effect completely overrides time discounting: an agent's payoff does not change with its discounting factor. In contrast, in our work we take a binary probability distribution over agent deadlines, but we consider this to be private knowledge. In addition to this, we also take a binary probability distribution over the price-surplus. But the agents do not know their opponent's bargaining cost or discounting factor. Our optimal strategies too give the entire surplus of price to the agent with the longer deadline. However because of the difference in initial offers, our results bring out the difference between the effect of deadlines and time discounting. That is, the deadline effect on payoffs to agents does not suppress the effect of time discounting.

3 The Negotiation Model

We use an alternating offers negotiation protocol for our study. Let b denote the buyer, s the seller and let $[P^a_{min}, P^a_{max}]$ denote the range of values for price that are acceptable to agent a, where $a \in \{b, s\}$. A value for price that is acceptable to both b and s, lies between their reservation prices, i.e., in the interval $[P^s_{min}, P^b_{max}]$. The difference between P^b_{max} and P^s_{min} is the *price-surplus*. Let $p^t_{b\to s}$ denote the price offered by agent b to agent s at time t. Negotiation starts when the first offer is made by an agent. When an agent, say s, receives an offer from agent b at time t, i.e., $p^t_{b\to s}$, it rates the offer using its utility function U^s. If the value of U^s for $p^t_{b\to s}$ at time t is greater than the value of the counter-offer agent s is ready to send at time t', i.e., $p^{t'}_{s\to b}$ with $t' > t$, then agent s accepts. Otherwise a counter-offer is made. Thus the action A that agent s takes at time t is defined as:

$$A^s(t', p^t_{b\to s}) = \begin{cases} Quit & \text{if } t > T^s \text{ where } T^s \text{ is the seller's deadline} \\ Accept & \text{if } U^s \text{ from } p^t_{b\to s} \geq U^s \text{ from counter-offer } p^{t'}_{s\to b} \\ p^{t'}_{s\to b} & \text{otherwise} \end{cases}$$

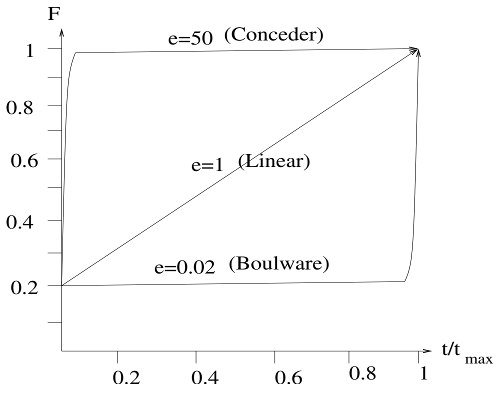


Fig. 1. Functions for the computation of $F(t)$

Given the fact that both agents have (different) deadlines, we assume that both agents use a strategy that varies their negotiation behaviour with respect to the passage of time. Thus, time is the predominant factor used to decide which value to offer in the next negotiation move. Here such strategies are called time-dependent (from [1]). Before determining the optimal strategies (in section 4), we briefly introduce the different types of time dependent strategies that we consider. These strategies vary the value of price depending on the remaining negotiation time, modeled as the above defined constant T^a. As in [1], the initial offer is a point in the interval $[P^a_{min}, P^a_{max}]$. Agents define a constant k^a that multiplied by the size of interval determines the price to be offered in the first proposal by agent a. Also the offer made by agent a to agent b at time t $(0 < t \leq T^a)$ is then modeled as a function F^a depending on time as follows:

$$p^t_{a \to b} = \begin{cases} P^a_{min} + F^a(t)(P^a_{max} - P^a_{min}) & \text{for the buyer} \\ P^a_{min} + (1 - F^a(t))(P^a_{max} - P^a_{min}) & \text{for the seller.} \end{cases}$$

A wide range of time dependent functions can be defined by varying the way in which $F^a(t)$ is computed. However, functions must ensure that $0 \leq F^a(t) \leq 1$, $F^a(0) = k^a$ and $F^a(T^a) = 1$. That is, the offer will always be between the value range, at the beginning it will give the initial constant and when the deadline is reached it will offer the reservation value. Function $F^a(t)$ is defined as follows [1]:

$$F^a(t) = k^a + (1 - k^a) \left(\frac{min(t, T^a)}{T^a} \right)^{\frac{1}{e}}$$

These families of functions represent an infinite number of possible strategies, one for each value of e. However, depending on the value of e, two extreme sets show clearly different patterns of behaviour (see Fig. 1):

1. *Boulware* [9]: For this strategy $e < 1$ and the initial offer is maintained till time is almost exhausted, when the agent concedes up to its reservation value.
2. *Conceder* [8]: For this strategy $e > 1$ and the agent goes to its reservation value very quickly. When $e = 1$ the price is increased linearly.

The value of a counter-offer depends on the initial price (IP) at which the agent starts negotiation, the final price (FP) beyond which the agent does not concede, e and T^a. Let V be a vector of these four variables, i.e., V = [IP, FP, e, T^a]. We call this the *counter-offer vector*. Let P^b denote P^b_{max}, P^s denote P^s_{min}, $P \in [P^b, P^s]$ and $T \in [0, T^a]$. The *negotiation outcome* O is an element of $\{(P, T), C\}$, where the pair (P, T) denotes the *price* and *time* at which agreement is reached and C denotes the conflict situation.

Agents' utility functions. The utility derived by agents depends on the final agreement on the price P and the duration of negotiation T. However, utility from price to an agent is independent of its utility from time, i.e., the buyer always prefers a low price and the seller always prefers a high price. Thus:

$$U^a : P \times T \rightarrow \Re \qquad a \in (b, s)$$

We consider the following two von Neumann-Morgenstern utility functions [4] as they incorporate the effects of discounting and bargaining costs:

1. *Additive form*:
$$U^a(P, T) = U^a_p(P) + U^a_t(T)$$
where U^a_p and U^a_t are unidimensional utility functions. This form is adequate when the only effect of time is a bargaining cost which is independent of the final outcome. We defined U^a_p as $U^b_p(P) = (P^b - P)$ for the buyer and $U^s_p(P) = (P - P^s)$ for the seller. U^a_t was defined as $U^a_t(T) = c^a T$. Thus when $(c^a > 0)$ the agent gains utility with time and when $(c^a < 0)$ the agent loses utility with time.
2. *Multiplicative form*:
$$U^a(P, T) = U^a_p(P) U^a_t(T)$$
where, as before, U^a_p and U^a_t are unidimensional utility functions. Here preferences for attribute P, given the other attribute T, do not depend on the level of T. This form is adequate when the effects of time are bargaining cost and discounting. U^a_p was defined as before and U^a_t was defined as $U^a_t(T) = (c^a)^T$. Thus when $(c^a > 1)$ the agent gains utility with time and when $(c^a < 1)$ the agent loses utility with time.

Agent a's utility from conflict is defined as $U^a(C) = 0$.

4 Optimal Negotiation Strategies

An agent's negotiation strategy defines the sequence of actions it takes during the course of negotiation. In our case, this equates to determining the value of a counter-offer which, in turn, depends on the counter-offer vector V. The information that an agent has about the negotiation parameters is called its *negotiation environment*. In order to determine an optimal strategy an agent needs to find values for V, on the basis of its negotiation environment, that maximize its utility. Since an agent's utility depends on two parameters, price and time (see section 3), it determines the optimal price P_o and the optimal time T_o for reaching an agreement. An optimal strategy thus makes counter-offers that result in the negotiation outcome (P_o, T_o).

4.1 Negotiation Environments

We model the negotiation environment E^b for an agent b, as a 9-tuple

$$\langle \mathcal{T}^s, \alpha^s, \alpha^s_c, \mathcal{P}^s, \beta^s, \beta^s_c, T^b, P^b, U^b \rangle$$

where:

- $\mathcal{T}^s$ denotes a two element vector that contains possible values for s's deadline such that the first element T^s_1 is less than the second element T^s_2;
- α^s denotes the probability that s's deadline is T^s_1 — the probability that it is T^s_2 is therefore $(1 - \alpha^s)$;
- α^s_c denotes the value of α^s on the basis of which the buyer selects an optimal strategy;
- $\mathcal{P}^s$ denotes a two element vector that contains possible values for s's reservation price such that the first element P^s_1 is less than the second element P^s_2;
- β^s denotes the probability that s's reservation price is P^s_1 — the probability that it is P^s_2 is therefore $(1 - \beta^s)$;
- β^s_c denotes the value of β^s on the basis of which the seller selects an optimal strategy;
- T^bdenotes the buyer's deadline;
- P^b denotes the buyer's reservation price;
- U^b denotes the buyer's utility which is a function of price and time that decreases with price and either increases or decreases with time.

Elements $\mathcal{T}^s, \alpha^s, \mathcal{P}^s$ and β^s represent the information that b has about s. This is the agent's private information. Our aim is to determine optimal strategies for agents by considering all possible ways in which time can effect negotiation. Thus depending on the type of utility function U^b and the relation between T^s_1, T^s_2 and T^b, the following six environments are defined:

E^b_1: When $T^s_2 < T^b$ and b gains utility with time, i.e., U^b is an increasing function of time.
E^b_2: When $T^s_1 < T^b < T^s_2$ and b gains utility with time.
E^b_3: When $T^s_1 > T^b$ and b gains utility with time.
E^b_4: When $T^s_2 < T^b$ and b loses utility with time, i.e., U^b is an decreasing function of time.
E^b_5: When $T^s_1 < T^b < T^s_2$ and b loses utility with time.
E^b_6: When $T^s_1 > T^b$ and b loses utility with time.

The environment for the seller can be defined analogously.

4.2 Negotiation Strategies

We formally define an agent's strategy (S^a) as a function that maps its negotiation environment, E^a, and time, T, to the counter-offer vector at time $(T + 1)$. Thus $S^a : (E^a, T) \rightarrow V^{T+1}$. Let O be the outcome that results from strategy S^a. S^a is the optimal strategy for agent a if it maximizes $U^a(O)$.

For any environment E^a_i, S^a_i denotes the corresponding optimal strategy. The optimal strategies are determined for each of the above six environments. An optimal strategy

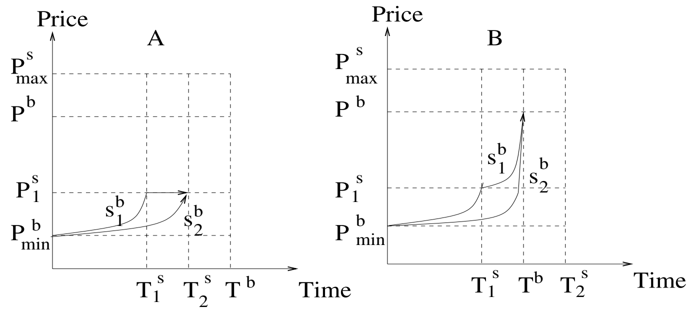


Fig. 2. Possible buyer strategies in environments E_1 and E_2

should result in agreement at the optimal price and at the optimal time. An agent's preference for price is independent of time, i.e., the buyer always prefers a low price and the seller always prefers a high price . We therefore determine the optimal strategy assuming $\beta^s = 1$ (section 4.3). This gives the optimal time for reaching agreement. Then for this optimal time we find the strategy that gives the optimal price by taking the actual value of β^s. This is explained in section 4.4. The resulting strategy is therefore optimal in both time and price.

4.3 Optimal Strategies in Particular Environments when $\beta^s = 1$

This section details the optimal strategy for each of the environments noted above. The analysis is from the perspective of the buyer, although strategies for the seller can be defined analogously.

Environment E_1^b. In this environment both the deadlines of s are less than T^b. As both agents use time dependent tactics, s will concede up to P_1^s latest by T_1^s or T_2^s. Hence it is enough if b uses some strategy that concedes only up to P_1^s. Since utility increases with time until T^b, b gets maximum utility if agreement is reached as late as possible. Since there are two possible values for the seller's deadline, negotiation could end either at T_1^s or T_2^s and there are two possible strategies b can use in this case (see Fig. 2A). It could use a strategy (s_1^b) that starts at P^b_{min} and approaches P_1^s at T_1^s and then makes no further concessions till T_2^s, or a strategy (s_2^b) that starts at P^b_{min} and approaches P_1^s at time T_2^s. These can be defined more formally as follows:

$s_1^b(E_1^b, T) = [P^b_{min}, P_1^s$, Boulware, $T_1^s]$ if $T \leq T_1^s$ and P_1^s otherwise and
$s_2^b(E_1^b, T) = [P^b_{min}, P_1^s$, Boulware, $T_2^s]$ for all values of T.

By using strategy s_1^b, b gets a utility of $U^b(P_1^s, T_1^s)$ with probability α^s and a utility of $U^b(P_1^s, T)$ with probability $(1-\alpha^s)$. Note that T lies between T_1^s and T_2^s depending on s's strategy. So the expected utility to b, EU^b, from strategy s_1^b is

$$EU_1^b = \alpha^s U^b(P_1^s, T_1^s) + (1-\alpha^s)U^b(P_1^s, T)$$

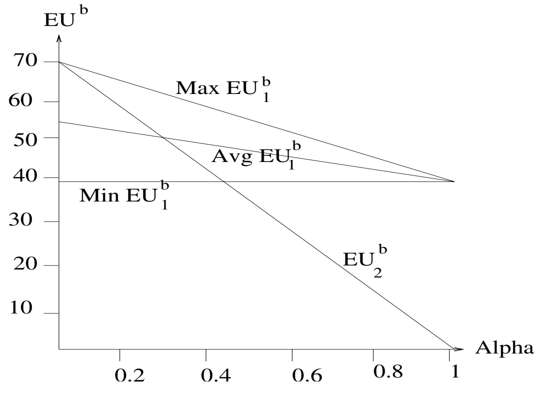


Fig. 3. Effect of α on the EU to buyer in E_1

where $T_1^s < T \le T_2^s$. On the other hand, if b uses strategy s_2^b, it gets a utility of $U^b(C)$ with probability α^s and a utility of $U_b(P_1^s, T_2^s)$ with probability $(1-\alpha^s)$. So its expected utility from strategy s_2^b becomes

$$EU_2^b = \alpha^s U^b(C) + (1-\alpha^s)U^b(P_1^s, T_2^s).$$

Turning now to an experimental evaluation of these strategies. Agent b's utility was defined as follows [2]

$$U^b(P,T) = \begin{cases} min(1, \frac{P^b-P}{P^b-P^b_{min}})T & \text{for } T \le T^b \\ 0 & \text{for } T > T^b \end{cases}$$

Thus b's utility linearly increases with price and time where $min(1, \frac{P^b-P}{P^b-P^b_{min}})$ gives a value between 0 and 1. The expected utility to b from any strategy depends on the following parameters: $[T_1^s, T_2^s, T^b, P_1^s, P^b, P^b_{min}\alpha^s]$. Let θ_t^s denote the length of time interval between T_1^s and T_2^s. For all contexts we assigned $T_1^s = 30$, $P^b = 70$, $P^b_{min} = 20$, $P_1^s = 30$ and $T^b = 100$. In our experiments we computed the expected utility for different values of θ_t^s. For each value of θ_t^s we varied α^s between 0 and 1 and found the EU^b in each case. In the first term for EU^b from strategy s_1^b, both parameters to U^b, i.e., P_1^s and T_1^s, are constants. In the second term, P_1^s is constant but T could be any value in the interval T_1^s to T_2^s depending on the strategy that the seller uses. As a result we get a range of values for EU^b from strategy s_1^b. Since utility to b is an increasing function of time, for the same price P_1^s, b gets maximum utility at T_2^s and minimum utility at T_1^s. All other possible values for EU^b lie within this range. We therefore computed the average EU^b for every value of α^s. All the parameters to U^b in the EU^b from strategy s_2^b are constants. This gives a single value for EU^b for every value of α^s.

[2] In all the experiments reported in this section we obtained similar results for additive and multiplicative forms of utility functions. We therefore describe here the experiments for only one multiplicative form.

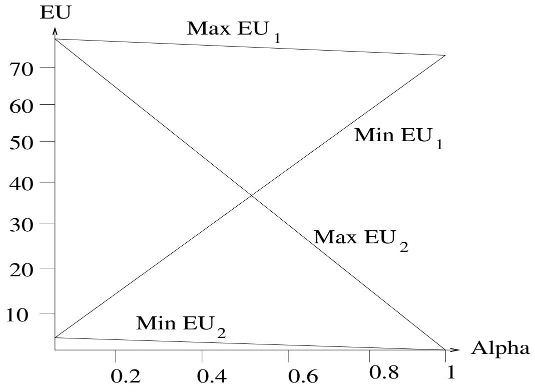


Fig. 4. Effect of α on the EU to buyer in E_2

The results of this experiment for $\theta^s_t = 40$ are as shown in Fig. 3. For values of α^s up to 0.3, strategy s^b_2 gives a higher EU^b than the average EU^b from strategy s^b_1. For higher values of α^s, the average EU^b from s^b_1 becomes higher than the EU^b from s^b_2. Let α^s_c denote the value of α^s below which s^b_2 is better than s^b_1 and above which s^b_1 is better than s^b_2. This value of α^s_c depends on the value of θ^s_t. As we increased θ^s_t, there was a corresponding increase in the value of α^s_c. A decrease in θ^s_t resulted in a corresponding decrease in α^s_c. Thus the optimal strategy S^b_1 is s^b_2 when $\alpha^s < \alpha^s_c$ and s^b_1 when $\alpha^s > \alpha^s_c$.

Environment E^b_2. As one of the possible deadlines for the seller, T^s_2, is greater than T^b, b needs to concede up to P^b in order to maximize its chances of reaching an agreement. Again it could use one of two possible strategies (see Fig. 2B). A strategy s^b_1 that starts at P^b_{min} and approaches P^b at T^b by crossing P^s_1 at T^s_1, or a strategy s^b_2 that starts at P^b_{min} and approaches P^b at T^b, crossing P^s_1 somewhere in the interval between T^s_1 and T^b. That is,

$$s^b_1(E^b_2, T) = \begin{cases} [P^b_{min}, P^s_1, \text{Boulware }, T^s_1] & \text{if } T \le T^s_1 \\ [P^s_1, P^b, \text{Boulware}, (T^b - T^s_1)] & \text{if } T > T^s_1 \end{cases}$$

$s^b_2(E^b_2, T) = [P^b_{min}, P^b, \text{Boulware}, T^b]$ for all values of T.

By using s^b_1, b gets a utility of $U^b(P^s_1, T^s_1)$ with probability α^s and utility $[\lambda U^b(P,T) + (1-\lambda)U^b(C)]$ with probability $(1-\alpha^s)$. λ denotes the probability that the seller crosses P^b prior to T^b. So the expected utility to b from strategy s^b_1 is

$$EU^b_1 = \alpha^s U^b(P^s_1, T^s_1) + (1-\alpha^s)[\lambda U^b(P,T) + (1-\lambda)U^b(C)]$$

where $(P^s_1 < P < P^b)$ and $(T^s_1 < T < T^b)$.

If b uses strategy s^b_2 it gets a utility of $U^b(C)$ with probability α^s and a utility of $[\lambda U^b(P,T) + (1-\lambda)U^b(C)]$ with probability $(1-\alpha^s)$ where $(P^s_1 < P < P^b)$ and $(T^s_1 < T < T^b)$. So b's expected utility from s^b_2 becomes

$$EU^b_2 = \alpha^s U^b(C) + (1-\alpha^s)[\lambda U^b(P,T) + (1-\lambda)U^b(C)]$$

In our experimental evaluation of these strategies, we get a range of values for EU^b from both strategies s^b_1 and s^b_2. We assigned $\lambda = 1$. The results of our experiments (for

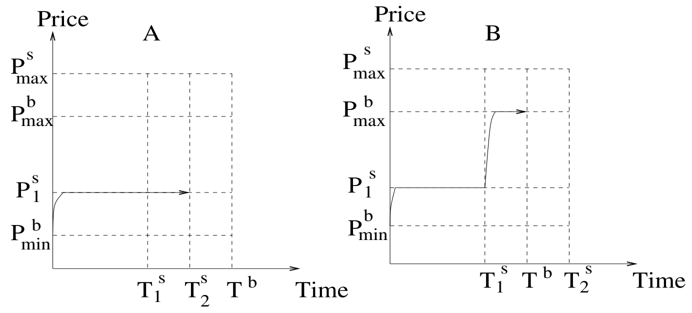


Fig. 5. Optimal buyer strategies when utility decreases with time

the same values of all other variables as in environment 1) are shown in Fig. 4. The average EU^b from strategy s^b_1 was higher than the average EU^b from s^b_2 for all values of α^s. Note that in this case the relationship between s^b_1 and s^b_2 does not depend on the value of θ^s_t. Thus the optimal strategy S^b_2 is s^b_1 for all values of α^s.

Environment E^b_3. Since both possible deadlines for the seller are greater than T^b, b has to use a strategy S^b_3 that starts at P^b_{min} and concedes up to P^b by T^b. Thus for all values of T

$$S^b_3(E^b_3, T) = [P^b_{min}, P^b, Boulware, T^b]$$

In the remaining three environments b's utility decreases with time, and it can maximize its utility by using a strategy that ends negotiation as early as possible.

Environment E^b_4. Here $T^s_2 < T^b$ and the buyer maximizes its utility by minimizing price by conceding only up to P^s_1 since it knows from its environment that s will offer P^s_1 latest by T^s_1 or T^s_2. In addition to this it can maximize its utility from time by offering P^s_1 as early as possible. Thus b's best strategy S^b_4 is to offer P^s_1 every time starting from its first offer (see Fig. 5A). Thus

$$S^b_4(E^b_4, T) = [P^b_{min}, P^s_1, Conceder, T^s_2]$$

for all values of T and the counter-offers are always P^s_1.

Environment E^b_5. In this environment b's best strategy is to offer P^s_1 from the start of negotiation till T^s_1. If T^s_1 is the actual deadline for the seller, negotiation would end latest by T^s_1 at price P^s_1. On the other hand, if T^s_2 is the seller's actual deadline, negotiation could continue beyond T^s_1 (depending on the seller's strategy). b now has to concede up to P^b at the end of T^s_1 since T^b is less than T^s_2. The strategy S^b_5 is shown in Fig. 5B. Thus

$$S^b_5(E^b_5, T) = \begin{cases} [P^b_{min}, P^s_1, Conceder, T^s_1] & \text{if } T \leq T^s_1 \\ [P^s_1, P_b, Conceder, (T^b - T^s_1)] & \text{if } T > T^s_1 \end{cases}$$

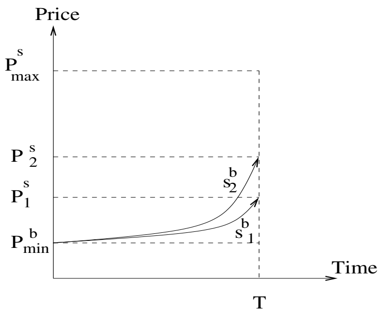


Fig. 6. Possible buyer strategies when $\beta \neq 1$

Environment E^b_6. In this environment b's best strategy S^b_6 is to concede up to P^b at the beginning of negotiation since $U^b(P^b, 1) \geq U^b(P^s, 2)$. Thus

$$S^b_6(E^b_6, T) = [P^b_{min}, P^b, Conceder, T^b]$$

for all values of T and the counter-offers are P^b throughout negotiation.

4.4 Optimal Strategies in Particular Environments when $\beta^s \neq 1$

The above strategies give an optimal value for T if $\beta^s = 1$. When $\beta^s \neq 1$ an optimal value for price needs to be determined. There are two possible values for the seller's reservation limit, P^s_1 with probability β^s and P^s_2 with probability $(1 - \beta^s)$. b now has two possible strategies (see Fig. 6). It can concede up to P^s_1 by T (strategy s^b_1) and get an EU^b of

$$EU^b_1 = \beta^s U^b(P^s_1, T) + (1 - \beta^s)U^b(C)$$

or concede up to P^s_2 by T (strategy s^b_2) and get an EU^b of

$$EU^b_2 = \beta^s U^b(P, T) + (1 - \beta^s)U^b(P^s_2, T)$$

where $(P^s_1 < P < P^s_2)$.

In terms of our experimental evaluation, the EU^b from any strategy depends on the following parameters: $[P^s_1, P^s_2, P^b, P^b_{min}, T, \beta^s]$. We assigned $P^s_1 = 30, P^s_2 = 50$. For all contexts we assigned $P^b = 70, P^b_{min} = 20, T = 50$ and $T^b = 100$. We varied β^s between 0 and 1 and computed the EU^b for both strategies for each value of β^s. The results of this experiment are as shown in Fig. 7. For values of β^s up to 0.65 the average utility from s^b_2 was higher than the EU^b from strategy s^b_1. For values of β^s higher than 0.65, the EU^b from strategy s^b_1 became higher than the average EU^b from strategy s^b_2. Let β^s_c denote the value of β^s below which s^b_2 is better than s^b_1 and above which s^b_1 is better than s^b_2. Let θ^p_s denote the difference between P^s_1 and P^s_2. The value of β^s_c depends on the value of θ^p_s. As we increased θ^p_s there was a corresponding decrease in the value of β^s_c. The decrease in θ^p_s resulted in a corresponding increase in β^s_c. Thus for $(\beta^s < \beta^s_c)$ the optimal price is P^s_2 and for $(\beta^s > \beta^s_c)$ the optimal price is P^s_1.

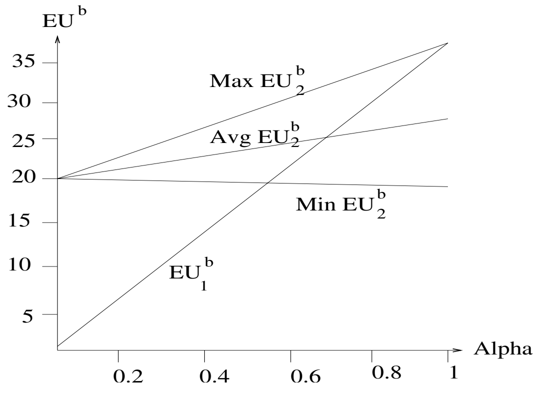


Fig. 7. Effect of β on the EU to buyer

Table 1. Optimal Strategies for the Buyer (P_o^s denotes the optimal value for the seller's reservation price)

Environment	Optimal Strategy
E_1^b	$[P_{min}^b, P_o^s$, Boulware, $T_2^s]$ for all values of T if $(\alpha^s < \alpha_c^s)$ $[P_{min}^b, P_o^s$, Boulware, $T_1^s]$ for all values of T if $(\alpha^s > \alpha_c^s)$
E_2^b	$[P_{min}^b, P_o^s$, Boulware, $T_1^s]$ if $T \leq T_1^s$ $[P_o^s, P^b$, Boulware, $(T^b$-$T_1^s)]$ if $T > T_1^s$
E_3^b	$[P_{min}^b, P^b$, Boulware, $T^b]$ for all values of T
E_4^b	$[P_{min}^b, P_o^s$, Conceder, $T_2^s]$ for all values of T
E_5^b	$[P_{min}^b, P_o^s$, Conceder, $T_1^s]$ if $T \leq T_1^s$ $[P_o^s, P^b$, Conceder, $(T^b$- $T_1^s)]$ if $T > T_1^s$
E_6^b	$[P_{min}^b, P^b$, Conceder, $T^b]$ for all values of T

5 Negotiation Outcomes

We now determine the outcome of negotiation when both agents use their respective optimal strategies as determined in table 1. Since the buyer and seller could either gain or lose utility with time, we have the following four possibilities:

1. Both buyer and seller gain utility with time.
2. Buyer gains and seller loses utility with time.
3. Buyer loses and seller gains utility with time.
4. Both buyer and seller lose utility with time.

The ordering on the deadlines of b and s will be one of the following:

1. $T_1^s < T_2^s < T_1^b < T_2^b$ (D1)
2. $T_1^b < T_2^b < T_1^s < T_2^s$ (D2)
3. $T_1^s < T_1^b < T_2^s < T_2^b$ (D3)
4. $T_1^s < T_1^b < T_2^b < T_2^s$ (D4)
5. $T_1^b < T_1^s < T_2^s < T_2^b$ (D5)
6. $T_1^b < T_1^s < T_2^b < T_2^s$ (D6)

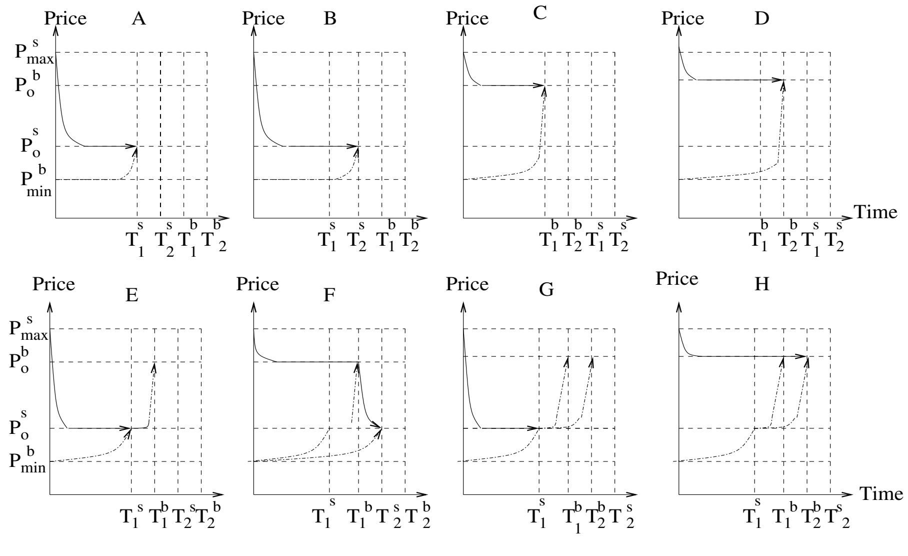


Fig. 8. Negotiation Outcome (thick lines denote seller strategy and dotted lines buyer strategy)

We consider the case where b gains with time and s loses with time and determine the outcome of negotiation for all possible orderings of deadlines. For ordering D1, let T^s_1 be s's actual deadline (see Fig. 8A). This corresponds to E^s_6 for the seller and its optimal strategy is S^s_6. For agent b it corresponds to E^b_1 (with a high value for α^s) irrespective of whether its deadline is T^b_1or T^b_2. Let P^s_o and P^b_o denote the optimal values for the seller's minimum reservation price and buyer's maximum reservation price respectively. Thus when b uses strategy S^b_1, s uses S^s_6, and the outcome of negotiation becomes (P^s_o, T^s_1). On the other hand, if T^s_2 is s's actual deadline (see Fig. 8B), it corresponds to E^s_6 for the seller and its optimal strategy is S^s_6. For the buyer it is E^b_1 and its optimal strategy is S^b_1 (α^s is low). The negotiation outcome now becomes (P^s_o, T^s_2). In both cases, since b's deadline is greater than s's deadline, it gets the entire surplus of price.

D2 is similar to D1 except that b and s are interchanged and the negotiation outcome is (P^b_o,T^b_1) when T^b_1 is b's actual deadline and (P^b_o, T^b_2) when T^b_2 is b's actual deadline (see Fig. 8C and 8D).

For D3, let T^s_1 be s's actual deadline (see Fig. 8E). This corresponds to E^s_6 for s and its optimal strategy is S^s_6. For the buyer it is E^b_2 if its deadline is T^b_1 and E^b_1 (with α^s high) if its deadline is T^b_2. Using the corresponding optimal strategies, in both cases the negotiation outcome is (P^s_o, T^s_1). If T^s_2 is s's actual deadline (see Fig. 8F), this corresponds to E^s_5 for s and its optimal strategy is S^s_5. Now if T^b_1 is b's actual deadline (E^b_2), its optimal strategy is S^b_2 and the negotiation outcome is (P^b_o,T^b_1); the entire surplus of price goes to S. But if $T^b_2$is b's actual deadline (E^b_1 with α^s low), its optimal strategy gives the outcome (P^s_o, T^s_2) and the entire surplus of price goes to b.

For D4, let T^s_1 be the seller's actual deadline (see Fig. 8 G). This corresponds to E^s_6 for the seller and its optimal strategy is S^s_6. If T^b_1 or T^b_2 is b's actual deadline (E^b_2), then its optimal strategy is S^b_2 in both cases. The negotiation outcome is (P^s_o, T^s_1). The entire

Table 2. Outcome of negotiation when both agents use their respective optimal strategies

Deadline Ordering	Seller's Deadline	Case 1, 2 and 3 Negotiation Outcome		Case 4 Negotiation Outcome	
		Buyer's Deadline		Buyer's Deadline	
D1	T_1^s	(P_o^s,T_1^s)	(P_o^s,T_1^s)	(P_o^s,T_0)	(P_o^s,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
	T_2^s	(P_o^s,T_2^s)	(P_o^s,T_2^s)	(P_o^s,T_0)	(P_o^s,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
D2	T_1^s	(P_o^b,T_1^b)	(P_o^b,T_2^b)	(P_o^b,T_0)	(P_o^b,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
	T_2^s	(P_o^b,T_1^b)	(P_o^b,T_2^b)	(P_o^b,T_0)	(P_o^b,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
D3	T_1^s	(P_o^s,T_1^s)	(P_o^s,T_1^s)	(P_o^s,T_0)	(P_o^s,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
	T_2^s	(P_o^b,T_1^b)	(P_o^s,T_2^s)	(P_o^b,T_0)	(P_o^s,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
D4	T_1^s	(P_o^s,T_1^s)	(P_o^s,T_1^s)	(P_o^s,T_0)	(P_o^s,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
	T_2^s	(P_o^b,T_1^b)	(P_o^b,T_2^b)	(P_o^b,T_0)	(P_o^b,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
D5	T_1^s	(P_o^b,T_1^b)	(P_o^s,T_1^s)	(P_o^b,T_0)	(P_o^s,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
	T_2^s	(P_o^b,T_1^b)	(P_o^s,T_2^s)	(P_o^b,T_0)	(P_o^s,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
D6	T_1^s	(P_o^b,T_1^b)	(P_o^s,T_1^s)	(P_o^b,T_0)	(P_o^s,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b
	T_2^s	(P_o^b,T_1^b)	(P_o^b,T_2^b)	(P_o^b,T_0)	(P_o^b,T_0)
		T_1^b	T_2^b	T_1^b	T_2^b

surplus of price goes to b. On the other hand, if T_2^s is s's actual deadline (E_4^b), its optimal strategy is S_4^b (see Fig. 8H).The optimal strategy for b is S_2^b irrespective of whether its actual deadline is T_1^b or T_2^b. The corresponding outcome of negotiation is (P_o^b, T_1^b) or (P_o^b, T_2^b), and the entire surplus goes to s. In the same way the outcome of negotiation, if both agents use the optimal strategies as determined in section 3, can be found for all the remaining cases. These results are summarized in table 2. T_0denotes the beginning of negotiation.

As seen from table 2, the negotiation outcome remains the same in the first three cases, i.e., when both agents gain utility with time, or when any one of them gains and the other loses with time. This happens because the agent that gains utility with time delays in making an offer that is acceptable to the opponent till the earliest deadline is reached. When both agents lose utility with time, they make the maximum required concession at the earliest opportunity, i.e. the start of negotiation. Agreement is therefore reached at the beginning of negotiation.

5.1 Conditions for Convergence of Optimal Strategies

The values of α_c^s and β_c^s are crucial in determining b's optimal strategy. For instance in negotiation environment E_1^b, the agent's optimal strategy S_1^b is s_2^b when $(\alpha^s < \alpha_c^s)$ and S_1^b is s_1^b when $(\alpha^s > \alpha_c^s)$. Thus in order for the strategies to converge, an agent, say b's, degree of belief α^s about s's deadline must be less than α_c^s when the actual deadline for s is T_2^s and it must be greater than α_c^s when the actual deadline for s is T_1^s.

Similarly the optimal value for price depends on the value of β_c^s. For instance, for the buyer, the optimal price is P_2^s if $(\beta^s < \beta_c^s)$ and it is P_1^s if $(\beta^s > \beta_c^s)$. Thus convergence is guaranteed only when the values of α^s and β^s satisfy the following conditions:

1. $(\alpha^s < \alpha_c^s)$ if $(T^s = T_2^s)$ and $(\alpha^s > \alpha_c^s)$ if $(T^s = T_1^s)$ for the buyer.
2. $(\alpha^b < \alpha_c^b)$ if $(T^b = T_2^b)$ and $(\alpha^b > \alpha_c^b)$ if $(T^b = T_1^b)$ for the seller.
3. $(\beta^s < \beta_c^s)$ if $(P^s = P_2^s)$ and $(\beta^s > \beta_c^s)$ if $(P^s = P_1^s)$ for the buyer.
4. $(\beta^b < \beta_c^b)$ if $(P^b = P_2^b)$ and $(\beta^b > \beta_c^b)$ if $(P^b = P_1^b)$ for the seller.

5.2 Payoffs to Agents

The utility that an outcome yields to an agent is a function of price and time. The factors that determine the outcome of negotiation are the agents' deadlines and their discounting or bargaining costs. We therefore analyze their effect on the utilities to both agents.

Effect of deadlines on the payoffs. As seen from table 2, when agents have unequal deadlines, the entire surplus of price goes to the agent with the longer deadline.

Effect of discounting/bargaining costs on payoffs. When at least one agent gains utility with time, agreement is reached at the earlier deadline; when both lose on time, agreement is reached toward the beginning of negotiation. Let the pair (U_b^1, U_s^1) denote the utility to b and s when both gain with time, (U_b^2, U_s^2) when b gains and s loses with time, (U_b^3, U_s^3) when b loses and s gains with time and (U_b^4, U_s^4) when both lose with time. When the effect of time on the two agents is not identical, i.e., when one of them gains utility with time and the other loses, the following result was observed. The agent that gains on time gets the same utility as compared to the case where both gain with time, i.e., for the buyer $(U_b^1 = U_b^2)$ and for the seller $(U_s^1 = U_s^3)$. The agent that loses on time gets less utility with time gets a lower utility when compared to the case where both gain or both lose with time, i.e., for the buyer $(U_b^3 < U_b^4)$ and $(U_b^3 < U_b^1)$ and for the seller $(U_s^2 < U_s^4)$ and $(U_s^2 < U_s^1)$.

6 Conclusions

This paper determined what the optimal negotiation strategies are for agents that find themselves in environments with different information states. Specifically, we considered situations where agents have uncertain information about two negotiation parameters (the opponent's deadline and reservation limit) but do not have any information about their opponent's bargaining cost, discounting factor or strategy. We listed conditions for convergence of these optimal strategies and studied the effect of time on the negotiation outcome. In the future we intend to extend our analysis to determine if this mutual strategic behavior leads to equilibria and then analyze situations where agents have limited information about other negotiation parameters like the opponent's bargaining cost, its discounting factor or its strategy to compare their relative influences on the negotiation outcome.

Acknowledgments

This work was supported by the EPSRC under grant GR/M07076 ("Practical Negotiation for Electronic Commerce").

References

1. P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1998.
2. J.C. Harsanyi and R. Selten. A generalized nash solution for two-person bargaining games with incomplete information. *Management Science*, 18(5):80–106, January 1972.
3. N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *Int Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.
4. R. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. New York: John Wiley, 1976.
5. S. Kraus, J. Wilkenfeld, and G. Zlotkin. Negotiation under time constraints. *Artificial Intelligence Journal*, 75(2):297–345, 1995.
6. Z. A. Livne. *The Role of Time in Negotiations*. PhD thesis, Massachusetts Institute of Technology, 1979.
7. A. Lomuscio, M. Wooldridge, and N. R. Jennings. A classification scheme for negotiation in electronic commerce. In F. Dignum and C. Sierra, editors, *Agent-Mediated Electronic Commerce: A European AgentLink Perspective*, pages 19–33. Springer Verlag, 2001.
8. D. G. Pruitt. *Negotiation Behavior*. Academic Press, 1981.
9. H. Raiffa. *The Art and Science of Negotiation*. Harvard University Press, Cambridge, USA, 1982.
10. J. S. Rosenchein and G. Zlotkin. *Rules of Encounter*. MIT Press, 1994.
11. A. Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, January 1982.
12. T. Sandholm and N. Vulkan. Bargianing with deadlines. In *AAAI-99*, pages 44–51, Orlando, FL, 1999.

Implicit Negotiation in Repeated Games

Michael L. Littman and Peter Stone

AT&T Labs Research
180 Park Avenue
Florham Park, NJ 07932-0971
{mlittman,pstone}@research.att.com

Abstract. In business-related interactions such as the on-going high-stakes FCC spectrum auctions, explicit communication among participants is regarded as collusion, and is therefore illegal. In this paper, we consider the possibility of autonomous agents engaging in implicit negotiation via their tacit interactions. In repeated general-sum games, our testbed for studying this type of interaction, an agent using a "best response" strategy maximizes its own payoff assuming its behavior has no effect on its opponent. This notion of best response requires some degree of learning to determine the fixed opponent behavior. Against an unchanging opponent, the best-response agent performs optimally, and can be thought of as a "follower," since it adapts to its opponent. However, pairing two best-response agents in a repeated game can result in suboptimal behavior. We demonstrate this suboptimality in several different games using variants of Q-learning as an example of a best-response strategy. We then examine two "leader" strategies that induce better performance from opponent followers via stubbornness and threats. These tactics are forms of implicit negotiation in that they aim to achieve a mutually beneficial outcome without using explicit communication outside of the game.

1 Introduction

In high-stakes, simultaneous, multicommodity auctions such as the ongoing FCC spectrum auctions (Weber 1997)[1], human bidders have been shown to bid strategically so as to threaten opponent bidders with retaliation in one market should the opponent compete in a different market. These strategic bids can be seen as implicit negotiation in a domain in which explicit communication is considered collusion, and is therefore illegal.

For such threats to work, the threatening agent must (i) communicate the intended retaliation it intends should the receiving agent not comply and (ii)

[1] The US Federal Communications Commission (FCC) holds spectrum auctions to sell radio bandwidth to telecommunications companies. Licenses entitle their owners to use a specified radio spectrum band within a specified geographical area, or *market*. Typically several licenses are auctioned off simultaneously with bidders placing independent bids for each license. The most recent auction, number 35, completed in January 2001 and brought in over $16 billion dollars.

convince the receiving agent that it is willing to execute the threat. Furthermore, the receiving agent must be able to understand that allowing the threat to be executed is not in its best interest. In FCC spectrum auctions, these threats and responses can be used to coordinate "strategic demand reduction," which can lead to substantial benefits to all participants. Weber (1997) described the importance of threats in these auctions as follows:

> What can sustain a tacit agreement among bidders concerning an allocation of licenses, when no binding agreements are legal? The force of threats can serve to stabilize an agreement. If two bidders have ceded licenses to one another, a subsequent attempt by one to violate the agreement can be immediately met with a response by the other, raising the prices of licenses held by the violator. (Weber 1997)

In this paper, we consider agents that negotiate by issuing and responding to threats in the context of repeated, bimatrix games (two-player, general-sum). In spite of their simplicity of form, bimatrix games present difficult challenges for agent learning and planning. Unlike in zero-sum games, where agents' objectives are diametrically opposed, agents participating in general-sum games can make concessions to their opponents and ultimately improve their own payoffs as a result. Thus, the behavior of the other agent becomes important, not just because of the damage it could cause, but for the benefits it can confer as an ally.

A standard approach to learning in games is to apply a "best response" strategy like Q-learning (e.g., Mundhe and Sen 2000). Q-learning has the benefit of being simple and providing the guarantee of an optimal response against a fixed opponent (Watkins and Dayan 1992). In a sense, best-response strategies like Q-learning are *followers* in that they attempt to maximize their own payoffs assuming their behavior has no effect on their opponents.

Against an unchanging opponent, the best-response agent learns to perform optimally. However, pairing two such followers in a repeated game can result in suboptimal behavior. In this paper, we explore simple strategies that act as *leaders* in that they behave in a way that maximizes their payoff factoring in the responsive behavior of the opposing agent. These leaders can be seen as engaging in implicit negotiation in the sense that they attempt to achieve a mutually beneficial outcome without the use of explicit communication outside of the game. To achieve this goal, a leader depends on the follower to collaborate, and it encourages such collaboration by making it in the follower's best interest. In the context of the Weber quote, the leaders are issuing threats and the followers are reacting to them.

In the next sections, we describe bimatrix games and Q-learning. Following that, we describe two leader strategies and then present experimental results in a number of simple games that show how leaders can improve payoffs for themselves and their followers.

2 Bimatrix Games

A bimatrix game is defined by a pair of matrices M_1 and M_2 of the same size (same number of rows and same number of columns). At each stage, the players choose actions, a row i for the row player and a column j for the column player. The row player receives payoff $M_1[i,j]$ and the column player receives payoff $M_2[i,j]$. The objective for the players is to maximize their average or discounted total payoff over an unbounded number of stages. For example, consider the following 2×2 game.

$$M_1 = \begin{bmatrix} a\ b \\ c\ d \end{bmatrix}, M_2 = \begin{bmatrix} e\ f \\ g\ h \end{bmatrix}.$$

If the row player selects action 0 and the column player selects action 1, then the row player receives payoff b and the column player receives payoff f.

To understand the intuitive connection between auctions and repeated games, imagine the following scenario. Each day two players engage in a simultaneous auction for two items, A and B. The bidding starts at \$1 and can go as high as \$3. Once a player drops out of the bidding for a particular item, it cannot place a later bid. Each player values each item at \$4. If a player bids for an item and the other does not, then the bidder will get it for \$1, leading to a net payoff of \$3. If both players continue bidding for the same item, the price will go up to \$3 and it is awarded randomly, leading to an expected payoff for each player of \$0.5. For this example, imagine that the row player can bid on item A or both items, while the column player can bid on item B or both items (allowing all combinations of bids does not change the example in a substantial way). Also imagine that the players are obstinate in that once deciding to bid for an item on a given day, they will continue bidding until their bid is declared a winner or the \$3 limit is reached (removing this assumption also does not change the example in a substantial way).

This scenario leads to the following payoffs where action 0 (top row or left column) represents bidding on just one item and action 1 represents bidding on both:

$$M_1 = \begin{bmatrix} \$3.0 & \$0.5 \\ \$3.5 & \$1 \end{bmatrix}, M_2 = \begin{bmatrix} \$3.0 & \$3.5 \\ \$0.5 & \$1 \end{bmatrix}.$$

If the other player bids for just one item, bidding for both items leads to the best possible payoff of \$3.5: \$3 from the uncontested item and \$0.5 from the contested item. However, if both bid for both items, they each expect only a \$1 payoff, whereas if they somehow coordinate their demand and each bid for one item (known as "demand reduction" in the auction literature), they can achieve a payoff of \$3 each. Alert readers will recognize this as a version of the game of the prisoner's dilemma.

A *behavior* or *strategy* in a bimatrix game specifies a method for choosing an action. In its most general form, a behavior specifies a probability distribution over action choices conditioned on the full history of past actions taken both by itself and by other agents.

One justifiable choice of behavior in a bimatrix game is for a player to maximize its payoff assuming the opponent will make this maximum as small as possible. This strategy can be called a minimax or security-level strategy. The *security level* is the expected payoff a player can guarantee itself using a minimax strategy. This strategy can be computed using linear programming (von Neumann and Morgenstern

In a *Nash equilibrium*, each player adopts a strategy that is a best response to the other—there is no incentive for unilateral deviation (Nash 1951). One shortcoming of two players adopting minimax strategies is that they need not be in Nash equilibrium. For example, consider the following game (called "chicken"):

$$M_1 = \begin{bmatrix} 3.0 & 1.5 \\ 3.5 & 1.0 \end{bmatrix}, M_2 = \begin{bmatrix} 3.0 & 3.5 \\ 1.5 & 1.0 \end{bmatrix}. \tag{1}$$

The minimax strategy is to always take action 0, since the agent can then get no worse than 1.5. However, both agents taking action 0 is not a Nash equilibrium, since either agent could improve by changing to action 1. (Either agent taking action 0 and the other taking action 1 is a Nash equilibrium in this game.)

In a *pareto-optimal behavior pair*, no player can improve its payoff without hurting the opponent. Another shortcoming of both players adopting the minimax strategy is that the result is not necessarily pareto-optimal: There might be other strategy pairs that are more beneficial to both parties. For example, consider the classic prisoner's dilemma:

$$M_1 = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}, M_2 = \begin{bmatrix} 3 & 5 \\ 0 & 1 \end{bmatrix}. \tag{2}$$

Here, the minimax strategy is to always take action 1. However, if both agents do so, their payoff will be 1. On the other hand, they can both do better if they somehow agree to both take action 0.

In this paper, we show that agents that can issue and respond to threats can, in effect, agree to play mutually beneficial strategies.

3 Q-learning

Q-learning is a reinforcement learning algorithm that is best justified for use in stationary, single-agent, fully observable environments (Markov decision processes or MDPs). However, it often performs well in environments that violate these assumptions. In its general form, a Q-learning agent can be in any state x of a finite set of states and can choose an action i from a finite set. It keeps a data structure $Q(x, i)$ that represents its expected payoff for starting in state x, taking action i, then behaving in a payoff-maximizing manner ever after. Each time the agent makes a transition from a state x to a state y via action i and receives payoff r, the Q table is updated according to

$$Q(x, i) = \alpha(r + \gamma \max_{i'} Q(y, i')) + (1 - \alpha)Q(x, i).$$

The parameters α and γ are both in the range 0 to 1. When the learning rate parameter α is close to 1, the Q table changes rapidly in response to new experience. When the discount rate γ is close to 1, future interactions play a substantial role in defining total payoff values.

In the repeated game context, there is a choice of what to use for the state-space of the learner. We studied two choices. The Q_0 approach uses just a single state (no state transitions). The Q_1 approach uses its action choice from the previous stage as its state.

Both agents choose actions according to the ϵ-greedy policy: In state x, choose

- a random action with probability ϵ
- $\text{argmax}_i\, Q(x, i)$ otherwise.

The random actions are exploration actions that give the learner an opportunity to find out if an action that looks less good may actually be better than its current preferred action choice.

Provided that its state space is expressive enough, Q-learning can learn a multi-step best response (best response against an agent can select actions conditioned on recent choices), since the multi-step best response problem is an MDP (Papadimitriou 1992).

4 Leader Strategies

This section describes two strategies—Bully and Godfather—that make action choices assuming that their opponents will be using a best response strategy such as one learned with Q-learning. They are general strategies that apply in all repeated bimatrix games and are based on concepts that generalize naturally to many competitive scenarios.

4.1 Bully

Bully is a deterministic, state-free policy that consistently plays action i^* defined by

$$i^* = \underset{i}{\text{argmax}}\, M_1(i, j_i^*)$$

where $j_i^* = \text{argmax}_j\, M_2(i, j)$. Here, M_1 is the leader's payoff matrix and M_2 is the follower's[2].

For example, in the game of chicken (Equation 1), the Bully behavior is to always choose action 1, since the opponent's best response to such a behavior is to always choose action 0, resulting in a payoff of 3.5 for Bully. Bully is an example of a Stackleberg leader (Fudenberg and Levine 1998).

The result of playing Bully against a follower is Nash-like: the follower is optimizing its payoffs assuming Bully stays fixed, and Bully has chosen to behave in the way that optimizes its payoffs assuming the other agent is a follower.

In a zero-sum game ($M_1 + M_2 = 0$), Bully is essentially a deterministic minimax strategy.

[2] If multiple values of j_i^* are possible due to ties in the matrix M_2, the safest thing is to assume the choice that leads to the smallest value of $M_1(i, j_i^*)$

4.2 Godfather

Godfather is a finite-state strategy that makes its opponent an offer it can't refuse. Call a pair of deterministic policies a *targetable pair* if playing them results in each player receiving more than its security level. Godfather chooses a targetable pair (if there is one) and plays its half (i.e., its action in the targetable pair) in the first stage. From then on, if the opponent plays its half of the targetable pair in one stage, Godfather plays its half in the next stage. Otherwise, it plays the policy that forces its opponent to achieve its security level. Thus, Godfather issues the threat: "Play your half of the targetable pair, or I'll force you to get no more than your security level no matter what you do."

Godfather is a generalization of tit-for-tat of prisoner's dilemma fame (Axelrod 1984). Note however that unlike in prisoner's dilemma, Godfather does not always select the best response to the opponent's defection. In games such as chicken, Godfather's punishment is chosen so as to minimize the opponent's payoff regardless even if it must settle for a low payoff itself. It is also a member of a more general class of finite-state strategies that uses the threat of a security-level outcome to maintain a mutually beneficial outcome. In a separate line of work, we have shown how to find Nash equilibria strategies of this form in polynomial time (Littman and Stone 2001). This result is in contrast to Nash equilibria in single stage bimatrix games, which are not known to be computable in polynomial time.

5 Experiments

To illustrate the importance of leader strategies, we compared Bully, Godfather, Q_0, and Q_1 in several different repeated games. Q_0 and Q_1 are used as representative best response strategies.

In our experiments, strategies Q_0 and Q_1 used the parameters $\gamma = 0.9$, $\alpha = 0.1$, and $\epsilon = 0.1$. Each experiment ran 30,000 stages, with the average payoff computed over the final 5,000 stages. Reported results reflect the mean and standard deviation over 100 experiments under identical conditions. Variance between experiments is due to stochastic strategies as well as random exploration in the strategies (ϵ).

All the games reported below are 2×2 bimatrix games with the diagonal payoffs of 3 (upper left) and 1 (lower right). We call action 0 "cooperate" and action 1 "defect". This makes "3" the mutual cooperation payoff and "1" the mutual defection payoff, in analogy with the prisoner's dilemma (Equation 2). In addition, $M_1^T = M_2$, so both players have the same payoff structure. By varying the off-diagonal payoffs, games with very different dynamics can be created.

The names of these games are in common usage in the game-theory community.

5.1 Deadlock: An Obvious Choice

Deadlock is a straightforward game in which, regardless of the opponent's choice, each player is better off cooperating:

$$M_1 = \begin{bmatrix} 3 & 2 \\ 0 & 1 \end{bmatrix}, M_2 = \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix}.$$

Bully chooses to cooperate in this game, and Godfather cooperates and uses defect as a threat.

The average payoffs (and standard deviations) for each strategy against Q_0 and Q_1 are:

	Q_0	Q_1	Bully	Godfather
Q_0	2.804 (0.008)	2.805 (0.009)	2.950 (0.003)	2.808 (0.011)
Q_1	2.805 (0.009)	2.803 (0.010)	2.950 (0.003)	2.805 (0.012)

Basically, all players cooperate and receive payoffs very close to that of mutual cooperation (3). Because of exploration, average payoffs are slightly lower.

5.2 Assurance: Suboptimal Preference

In the assurance game, it is more important to match the other player's choice than it is to cooperate:

$$M_1 = \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix}, M_2 = \begin{bmatrix} 3 & 2 \\ 0 & 1 \end{bmatrix}.$$

Thus, if the chance that the opponent will defect is more than 50%, it is better to defect. It is also better to defect from a minimax perspective, making it the "safest" alternative.

In this game, two Q learners will typically coordinate their choices, but with no particular bias as to which coordination point is chosen. As a result, the expected score is less than $(1+3)/2 = 2$ with a high variance. Thus, a pair of Q learners perform suboptimally in this game:

	Q_0	Q_1	Bully	Godfather
Q_0	1.431 (0.760)	1.537 (0.813)	2.850 (0.009)	1.387 (0.683)
Q_1	1.927 (0.886)	1.662 (0.846)	2.850 (0.009)	2.805 (0.010)

Bully, by steadfastly choosing to cooperate, invites the learners to cooperate and achieves the maximum score. Similarly, Godfather, by threatening defection each time its opponent defects, teaches Q_1 that mutual cooperation is its best response. Godfather is not able to teach Q_0 this lesson, since Q_0 cannot remember its previous action—the one being rewarded or punished.

It is worth noting that changing the value “2” in the payoff matrices changes the probability that a pair of Q learners will cooperate. As the value decreases, the mutual cooperation equilibrium becomes easier to find for Q learners, in part because the expected payoff for defecting against a random opponent decreases. On the other hand, as the value increases, finding the mutual cooperation equilibrium becomes harder. Indeed, if the value exceeds the mutual cooperation payoff (3), the game changes into the prisoner’s dilemma, described next.

5.3 Prisoner’s Dilemma: Incentive to Defect

Regardless of the opponent’s choice, a player in the prisoner’s dilemma is better off defecting:

$$M_1 = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}, M_2 = \begin{bmatrix} 3 & 5 \\ 0 & 1 \end{bmatrix}.$$

As in “deadlock”, Q learners are sensitive to the dominance of one choice over the other and quickly converge. In this case, however, the payoff is suboptimal.

Bully’s strategy in this game is to defect. The Godfather strategy is tit-for-tat: cooperate if the opponent cooperates and defect otherwise. Here, we get the following results:

	Q_0	Q_1	Bully	Godfather
Q_0	1.179 (0.115)	1.156 (0.028)	1.202 (0.011)	1.383 (0.324)
Q_1	1.169 (0.038)	1.204 (0.085)	1.198 (0.010)	2.947 (0.004)

The combination of Godfather and Q_1 is the only pair that achieves the mutual cooperation payoff in this game. Interestingly, Godfather appears to be able to sometimes (high variance) lure Q_0 to the higher paying policy.

5.4 Chicken: Incentive to Exploit

In chicken, each player is better off choosing *the opposite* action of its opponent:

$$M_1 = \begin{bmatrix} 3.0 & 1.5 \\ 3.5 & 1.0 \end{bmatrix}, M_2 = \begin{bmatrix} 3.0 & 3.5 \\ 1.5 & 1.0 \end{bmatrix}.$$

The game is suggestive of the “game” of highway chicken, in which two cars approach each other at high speed. At the last moment, the drivers must decide whether to veer away (cooperate) or keep going straight (defect). If one driver veers and the other doesn’t, the “chicken” is given a low payoff and the other player gets a high payoff for bravery. However, if neither player chickens out, the result is an extremely low score for both.

Chicken can be a trickier game to reason about than the prisoner’s dilemma. From a minimax perspective, the best strategy is to cooperate. However, if a player notices that its opponent consistently cooperates, it has an incentive to exploit this fact by defecting. Once one of the players defect, the result is stable.

In chicken, there is an incentive to act stupid—if you can convince your opponent that you will defect, no matter what, your opponent will eventually back down and cooperate, maximizing your score. Two smart opponents will each try to convince the other that it won't back down, resulting in a kind of meta-game of chicken (hold off learning until the last possible moment). In addition, whereas one could argue that, in prisoner's dilemma, tit-for-tat is the best either player can reasonably expect to score (there is no way to imagine inducing an opponent to repeatedly accept the 0 "sucker" payoff), tit-for-tat in chicken is marginally less attractive than the stable option of defecting against the opponent's cooperation.

Our results illustrate the complexity of the game:

	Q_0	Q_1	Bully	Godfather
Q_0	2.452 (0.703)	2.535 (0.527)	3.375 (0.007)	2.849 (0.010)
Q_1	2.391 (0.443)	2.868 (0.015)	3.374 (0.007)	2.948 (0.004)

The most successful strategy in this game is Bully, which repeatedly defects and waits for the learner to cooperate in response. Once again, the Godfather–Q_1 combination is able to find the mutual cooperation payoff. Not surprisingly, most combinations of follower vs. follower result in a payoff of approximately $(1.5 + 3.5)/2 = 2.5$ with high variance as the learners randomly choose one of the two asymmetric equilibria.

Two surprises are the results for the combinations of Q_1–Q_1 and of Q_0–Godfather. The low variance and the high score suggests that these combinations consistently settle on mutual cooperation. This cooperation is surprising since Q_1 cannot represent the threatening strategy and Q_0 cannot respond to it. In fact, mutual cooperation is *not* completely stable for these players. Preliminary investigation indicates that the Q-learning agents are learning to cooperate temporarily, but then periodically exploring the result of defection since they are unable to remember the negative effect of doing so. Each time they defect, it works out well at first. But eventually, as the opponent re-adapts or punishes the defecting agent, the agent "remembers" why it is better to cooperate and switches back to cooperation for a period of time before forgetting and exploring once again. This cycle repeats on the order of every few dozen stages.

6 Related Work

There is a huge literature on repeated games, equilibria and learning; introductory textbooks on game theory (Osborne and Rubinstein 1994) serve as an entryway to this set of ideas. The most relevant research to our current work is on "Folk Theorems." To a first approximation, folk theorems show that there are strategies, like Godfather, that support mutually desirable outcomes, like our targetable pairs, in a Nash equilibrium sense. Our focus is on the use of these strategies as counterparts to more direct best response strategies in computational experiments.

Our work grows out of recent attempts to use game theory as an underpinning for multiagent reinforcement learning. Unlike Hu and Wellman (1998) and Littman (2001), our work examines learning in general-sum games that may include neither adversarial nor coordination equilibria. Unlike Greenwald et al. (2001), Jafari et al. (2001), Claus and Boutilier (1998), and others, we looked at combining best response strategies with "leader" strategies. Note that the strategies we explored need not be Nash in the strict sense. This is because it is very difficult to compute a best response to an opponent whose strategy is the Q-learning algorithm. For example, in the Prisoner's dilemma, there might be a strategy that performs better than tit-for-tat against Q_0 by first "teaching" Q_0 to cooperate, then exploiting it for several steps before returning to cooperation mode. Constructing such a strategy requires detailed knowledge of the parameters used in the learning algorithm and the precise update rule used.

With regards to inter-agent negotiation, traditional methods rely on communication among participating agents (e.g., Sandholm and Lesser 1995, Zlotkin and Rosensc
Coordination without communication generally relies upon common knowledge or preferences that is either pre-programmed into the agents or assumed to exist a priori (e.g., Fenster et al. 1995, Stone and Veloso 1999). In contrast, this work examines the possibility of coordination via repeated interactions.

7 Conclusions

This paper illustrates the importance of strategies that can lead best-response agents in repeated games. We showed that the combination of two basic Q-learners (Q_0–Q_0) results in suboptimal payoffs in 3 out of the 4 games studied. We described a simple stationary leading strategy, Bully, and a more complex 2-state strategy, Godfather. Both Bully and Godfather are general strategies that apply across a range of games. Godfather attempts to stabilize a mutually beneficial payoff by punishing its opponent whenever it deviates from its assigned action. We showed that a 2-state Q-learner (Q_1) that remembers its immediately previous action learns to respond consistently to Godfather's threats. We conclude that (a) agents need to go beyond straight best response to succeed in a broad range of scenarios, and (b) it is not necessary to resort to complicated strategies to do so.

In our experiments, the combination of Godfather–Q_1 was the only one that settled on mutual cooperation in every game. This combination led to the highest score attained in all games except chicken, in which Bully was able to overpower the learning agents to achieve a higher score. However, even in chicken, Godfather might be a safer strategy, since Godfather–Godfather would achieve mutual cooperation, whereas Bully–Bully would result in mutual defection.

Indeed, although we have not presented results of our leader strategies playing against themselves, it is the case that it can result in suboptimal performance due to the fact that they assume that their opponents can adapt to their strategies. Rather, the most successful pairings, as explored in this paper, are between a leader and a follower.

Faced with an unknown opponent, it is not clear which role an agent should adopt. A natural approach would be to mix leader-like qualities and follower-type qualities as a function of the opponent's behavior. Human agents in high-stakes multicommodity auctions, for example, have been shown to both issue and respond to threats as the need arises (Cramton and Schwartz 2000).

In any case, the results presented in this paper suggest that when an agent is interacting with other agents in a competitive game-like environment, it should consider whether the other agents are using a best response strategy. If so, the agent should look for ways to apply the Bully and/or Godfather strategy. Meanwhile, if credible, there is a potential advantage in some situations to convincing a leader that the agent is not capable of recognizing the effects of its actions sufficiently to be worth threatening.

Our on-going research agenda involves creating agents that can implicitly negotiate by issuing and responding to threats in a detailed simulator of the FCC spectrum auctions (Csirik et al. 2001, Reitsma et al. 2002). It is clear that an agent can effectively threaten opponents provided that that the opponents are able to understand the threats. However, it is not so clear what mechanism underlies the opponents' response to these threats. Nonetheless, human bidders clearly show behaviors of both issuing and responding to threats. In this work, we have taken a first step toward understanding how to implement agents that can perform this important type of negotiation.

Acknowledgements

We thank Avi Pfeffer, Amy Greenwald, Craig Boutilier, and Jordan Pollack and his research group for helpful discussions.

References

1. Robert Axelrod. *The Evolution of Cooperation.* Basic Books, 1984.
2. Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752, 1998.
3. Peter Cramton and Jesse Schwartz. Collusive bidding: Lessons from the FCC spectrum auctions. *Journal of Regulatory Economics*, 17:229–252, 2000.
4. János A. Csirik, Michael L. Littman, Satinder Singh, and Peter Stone. FAucS: An FCC spectrum auction simulator for autonomous bidding agents. In Ludger Fiege, Gero Mühl, and Uwe Wilhelm, editors, *Electronic Commerce: Proceedings of the Second International Workshop*, pages 139–151, Heidelberg, Germany, 2001. Springer Verlag.
5. Maier Fenster, Sarit Kraus, and Jeffrey S. Rosenschein. Coordination without communication: Experimental validation of focal point techniques. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 102–108, Menlo Park, California, June 1995. AAAI Press.
6. Drew Fudenberg and David K. Levine. *The Theory of Learning in Games.* The MIT Press, 1998.

7. Amy Greenwald, Eric Friedman, and Scott Shenker. Learning in network contexts: Experimental results from simulations. *Journal of Games and Economic Behavior*, 35(1/2):80–123, 2001.
8. Junling Hu and Michael P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In Jude Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250, 1998.
9. Amir Jafari, Amy Greenwald, David Gondek, and Gunes Ercal. On no-regret learning, fictitious play, and Nash equilibria. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 226–233. Morgan Kaufmann, 2001.
10. Michael L. Littman. Friend-or-foe Q-learning in general-sum games. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 322–328. Morgan Kaufmann, 2001.
11. Michael L. Littman and Peter Stone. A polynomial-time algorithm for computing Nash equilibria in repeated bimatrix games. Research note, 2001.
12. Manisha Mundhe and Sandip Sen. Evaluating concurrent reinforcement learners. In *Proceedings of the Fourth International Conference on Multiagent Systems*, pages 421–422. IEEE Press, 2000.
13. J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.
14. Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.* The MIT Press, 1994.
15. Christos H. Papadimitriou. On players with a bounded number of states. *Games and Economic Behavior*, 4:122–131, 1992.
16. Paul S. A. Reitsma, Peter Stone, Janos A. Csirik, and Michael L. Littman. Randomized strategic demand reduction: Getting more by asking for less. Submitted, 2002.
17. Tuomas Sandholm and Victor Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 328–335, Menlo Park, California, June 1995. AAAI Press.
18. Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, June 1999.
19. J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, NJ, 1947.
20. Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8 (3):279–292, 1992.
21. Robert J. Weber. Making more from less: Strategic demand reduction in the FCC spectrum auctions. *Journal of Economics and Management Strategy*, 6(3):529–548, 1997.
22. Gilad Zlotkin and Jeffrey S. Rosenschein. Mechanisms for automated negotiation in state oriented domains. *Journal of Artificial Intelligence Research*, 5:163–238, 1996.

Dialogues for Negotiation: Agent Varieties and Dialogue Sequences

Fariba Sadri[1], Francesca Toni[1], and Paolo Torroni[2]

[1] Department of Computing, Imperial College,
180 Queens Gate, SW7 London, UK
{fs,ft}@doc.ic.ac.uk
[2] DEIS, Università di Bologna
V.le Risorgimento 2, 40136 Bologna, Italy
ptorroni@deis.unibo.it

Abstract. This work presents a formal, logic-based approach to one-to-one agent negotiation, in the context of goal achievement in systems of agents with limited resource availability. The proposed solution is based on agent dialogues as a way, e.g., to request resources and propose resource exchanges. It relies upon agents agreeing solely upon a language for negotiation, while possibly adopting different negotiation policies, each corresponding to an agent variety. Agent dialogues can be connected within sequences, all aimed at achieving an individual agent's goal. Sets of sequences aim at allowing all agents in the system to achieve their goals, and can be used, e.g., to solve the resource reallocation problem. We model dialogues via logic-based dialogue constraints, and propose an execution model to be adopted for the automatic generation of dialogues. We also propose an agent dialogue cycle for the automatic generation of sequences of dialogues. Finally, we identify some desirable properties of dialogues, dialogue sequences and sets of such sequences, and prove/disprove these properties for concrete agent varieties.

1 Introduction

Negotiation is one of the main research areas in multi-agent systems. In many cases agents need to negotiate because they operate in environments with limited resource availability. The area of negotiation is broad and applies to different scenarios [5]. For example, *one-to-many* negotiation is used for auctions, where auctioneer and bidders reach an agreement on the cost of the items on sale. Most research in the area of one-to-many negotiation adopts a game-theoretic approach, with interesting results but under the often unrealistic assumptions that the computational resources of agents are unlimited and that their knowledge of the world is global and complete. Another example of negotiation is *one-to-one* negotiation, used, for instance, for task reallocation [15] and for resource reallocation [11], where the limited resources may be time, the computational resources of agents, or physical resources needed to carry out some tasks. Many approaches in the area of one-to-one negotiation are heuristic-based and, in spite of their experimentally proven effectiveness, they do not easily lend themselves to expressing theoretically provable properties. Other approaches present a good descriptive model, but fail to provide an execution model that can help to forecast the behaviour of any corresponding implemented system.

In this paper we tackle one-to-one negotiation between agents that share the same language for negotiation but may adopt different negotiation policies. We approach the problem in a logic-based manner that allows for proving properties such as correctness and completeness. We also propose an execution model to be used in order to implement the system. Although we concentrate on the specific problem of resource reallocation, our approach is more general, as far as the representation of policies and the reasoning that generates dialogues and their sequences, and could be extended to be applicable to other negotiation domains.

The framework we propose is based on agent dialogues, as a way to request resources, propose resource exchanges, suggest alternative resources, etc. We assume that the agents are equipped with a planner that, given a goal, is able to produce a sequence of actions to achieve it and an associated set of missing resources that are needed to carry out the plan. A single dialogue is used to obtain a resource. We describe an agent dialogue cycle that can be used to trigger a new dialogue after one is terminated, in order to allow the agent to collect all the resources needed to achieve its goal. In this way, the agents of a system produce sequences of dialogues, and in general sets of sequences of dialogues (a sequence for each agent that needs to obtain any missing resources).

We study some properties of dialogues, sequences of dialogues, and sets of sequences of dialogues in an agent system. We define the properties of termination, correctness and completeness, and show whether they hold for some concrete agent varieties we propose. Finally, we study how a heterogeneous system, i.e., a system composed by a variety of agents, can be used to implement resource reallocation policies. Note that we do not make any concrete assumption on the internal structure of agents, except for requiring that they hold beliefs, goals, intentions and, possibly, resources.

2 Preliminaries

Capital and lower-case letters stand for variables and ground terms, respectively.

Definition 1 (*Performative or dialogue move*)
A *performative* or *dialogue move* is an instance of a schema $tell(X, Y, \mathbf{Subject}, T)$, where X is the *utterer* and Y is the *receiver* of the performative, and T is the *time* when the performative is uttered. $\mathbf{Subject}$ is the *content* of the performative, expressed in some given *content language*.

A concrete example of a performative is $tell(a, b, \mathbf{request}(\mathbf{give}(nail)), 1)$, where $\mathbf{Subject}$ is $\mathbf{request}(\mathbf{give}(nail))$. Intuitively, this performative represents a's request to b for a nail, at time 1. We will often refer to a performative simply as $p(T)$, if we want to emphasise the time of the performative, or p.

Definition 2 (*Language for negotiation*)
A *language for negotiation* $\mathcal{L}$ is a (possibly infinite) set of (possibly non ground) performatives. For a given $\mathcal{L}$, we define two (possibly infinite) subsets of performatives, $\mathcal{I}(\mathcal{L})$, $\mathcal{F}(\mathcal{L}) \subseteq \mathcal{L}$, called respectively *initial moves* and *final moves*. Each final move is either *successful* or *unsuccessful*.

The following is a possible concrete language for negotiation:

$\mathcal{L}_1$ = { (1) $\mathit{tell}(X, Y, \mathbf{request}(\mathbf{give}(\mathit{Resource})), T)$
(2) $\mathit{tell}(X, Y, \mathbf{accept}(\mathit{Move}), T)$
(3) $\mathit{tell}(X, Y, \mathbf{refuse}(\mathit{Move}), T)$
(4) $\mathit{tell}(X, Y, \mathbf{challenge}(\mathit{Move}), T)$
(5) $\mathit{tell}(X, Y, \mathbf{justify}(\mathit{Move}, \mathit{Support}), T)$
(6) $\mathit{tell}(X, Y, \mathbf{promise}(\mathbf{give}(\mathit{Resource}), \mathbf{give}(\mathit{Resource}')), T)$ }

where (4) is used by X to ask Y a reason (justification) for a past Move, (5) is used by X to justify to Y a past Move by means of a $\mathit{Support}$, (6) is used by X to propose a deal: X will give $\mathit{Resource}'$ to Y if Y will give $\mathit{Resource}$ to X (see Sadri et al. [14] for a more detailed description). The initial and final moves are:

$\mathcal{I}(\mathcal{L}_1) = \{\mathit{tell}(X, Y, \mathbf{request}(\mathbf{give}(\mathit{Resource})), T)\}$
$\mathcal{F}(\mathcal{L}_1) = \{$ [*successful moves:*]
$\mathit{tell}(X, Y, \mathbf{accept}(\mathbf{request}(\mathbf{give}(\mathit{Resource}))), T)$,
$\mathit{tell}(X, Y, \mathbf{accept}(\mathbf{promise}(\mathbf{give}(\mathit{Resource}), \mathbf{give}(\mathit{Resource}'))), T)$,
[*unsuccessful moves:*]
$\mathit{tell}(X, Y, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(\mathit{Resource}))), T)$,
$\mathit{tell}(X, Y, \mathbf{refuse}(\mathbf{promise}(\mathbf{give}(\mathit{Resource}), \mathbf{give}(\mathit{Resource}'))), T)\}$.

As in this paper we are interested in negotiation for the exchange of resources, we will assume that there always exists a **request** move in the (initial moves of any) language for negotiation.

Definition 3 (*Agent system*)
An *agent system* is a finite set A, where each $x \in A$ is a ground term, representing the name of an agent, equipped with a *knowledge base* $\mathcal{K}(x)$.

We will assume that, in an agent system, the agents share a common language for negotiation as well as a common content language.

In this paper, we presume a logic language for the knowledge base of agents, but make only three assumptions on the syntax of this language. In particular, we assume that it contains notions of *literal*, *complement* of sentences, *conjunction* of sentences, *true* and *false*, and that such language is equipped with a notion of *entailment*, such that, for every ground literal in the language, either the literal or its complement is entailed from the set of sentences that represent the knowledge of agents, and such that no literal and its complement are entailed at the same time. We also make concrete assumptions on the syntax of a component of the knowledge base, consisting of "dialogue constraints" (described in Section 3).

As far as the content of the knowledge base, we will assume that it describes, in addition to the dialogue constraints,

– domain-specific as well as domain-independent beliefs that the agent can use to generate *plans* for its *goal*, such as the knowledge describing preconditions and effects of actions. We will not make any assumptions on how such plans are generated; in particular, they could be obtained from a planner, either within the agent or external to it, or be extracted from a given library of plans, which is part of the knowledge of the agent;

- information about the resources available to the agent;
- information about the dialogues in which the agent has taken part;
- information on the selected *intention*, consisting of the given *goal*, a *plan* for the given goal, as well as the set of resources already *available* to the agent and the set of *missing* resources, both required for the plan to be executable.

Note that we assume that each agent is equipped with a *single (possibly conjunctive) goal*. For the sake of simplicity, we will also assume that this goal does not change during the dialogues. Such assumption could be relaxed, and our framework extended, for example by letting the agents' modify their goals after they have been achieved, or proven impossible to achieve, in accordance with the Rao and Georgeff's BDI theory [12] that requires goals to be believed feasible.

The purpose of negotiation is for the agent to obtain the missing resources, while retaining the available ones that are necessary for the plan in its current intention.

We do not make any assumptions on how the plan within an intention is obtained. However, we assume that the intention is such that the plan it contains allows the goal to be achieved, with respect to the knowledge base of the agent. We also assume that the missing resources in the intention and those available to the agent, as given by the knowledge base, have no intersection, and that the available resources in the intention are contained in the resources available to the agent, as given by the knowledge base.

In the sequel, we will refer to the knowledge base, intention and goal of an agent x as $\mathcal{K}(x)$, $\mathcal{I}(x)$ and $\mathcal{G}(x)$, respectively, or simply as $\mathcal{K}$, $\mathcal{I}$ and $\mathcal{G}$, respectively, if clear from the context.

As an example, for some agent $x \in A$, the knowledge base $\mathcal{K}(x)$ might contain beliefs such as (0 stands for the initial time)
$have(picture, 0)$, $have(hammer, 0)$, $have(screwdriver, 0)$;
the goal $\mathcal{G}(x)$ might be $hung(picture)$, and the intention $\mathcal{I}(x)$ might be
$\{\ goal(\{hung(picture)\}),$
$plan(\{obtain(nail), hit(nail), hang(picture)\}, 0),$
$available(\{hammer, picture\}, 0),$
$missing(\{nail\}, 0)\ \}.$
This example is inspired by Parsons et al. [11]. There, two agents need to hang some objects (a picture the one, a mirror the other). Each agent could either use a hammer and a nail or a screw and a screwdriver, but neither agent owns both resources, hence the need for negotiation. The agent's knowledge is time-stamped as it might change over time, e.g., by acquisition/loss of resources. Moreover, as we will see in the remainder of the paper, the intention might also change as a result of negotiation, when resources are exchanged. For instance, after two agents, negotiating with the language $\mathcal{L}_1$, agree on a promise, both agents' knowledge bases will be different, since they will have different resources, and possibly different plans. Finally, note that we assume that actions within plans are implicitly temporally ordered (left-to-right). For simplicity, we do not address the issue of the time of actions and goals in this paper. For a more concrete description of the knowledge base of agents see [14].

In this paper, we will assume that (any ground instance of) the literal $have(R, T)$ holds (namely it is entailed by $\mathcal{K}(x)$, for the given agent x) if the agent has the resource R at the time T of interest, and (ground instances of) the literal $need(R, T)$ holds if the agent's intention identifies R as $available$.

In the sequel, given any intention $\mathcal{I}$,

- $goal(G) \in \mathcal{I}$ will stand for 'G is the goal in $\mathcal{I}$'
- $plan(P, T) \in \mathcal{I}$ will stand for 'P is the plan in $\mathcal{I}$, time-stamped with time T'
- $missing(Rs, T) \in \mathcal{I}$ will stand for 'Rs are the resources needed to make the plan in $\mathcal{I}$ executable but currently missing, time-stamped with time T'
- $available(Rs, T) \in \mathcal{I}$ will stand for 'Rs are the resources needed to make the plan in $\mathcal{I}$ executable and currently available, time-stamped with time T'.

We will omit the time T if clear from the context.

3 Dialogues

For a given agent $x \in A$, where A is equipped with $\mathcal{L}$, we define the sets

- $\mathcal{L}^{in}(x)$, of all performative schemata of which x is the receiver, but not the utterer;
- $\mathcal{L}^{out}(x)$, of all performative schemata of which x is the utterer, but not the receiver.

Note that we do not allow for agents to utter performatives to themselves. In the sequel, we will often omit x, if clear from the context, and simply write $\mathcal{L}^{in}$ and $\mathcal{L}^{out}$.

Negotiation policies can be specified by sets of dialogue constraints:

Definition 4 (*Dialogue constraint*)
Given an agent system A, equipped with a language for negotiation $\mathcal{L}$, and an agent $x \in A$, a *dialogue constraint* for x is a (possibly non-ground) if-then rule of the form: $p(T) \wedge C \Rightarrow \hat{p}(T+1)$, where

- $p(T) \in \mathcal{L}^{in}(x)$ and $\hat{p}(T+1) \in \mathcal{L}^{out}(x)$,
- the utterer of $p(T)$ is the receiver of $\hat{p}(T+1)$, and the receiver of $p(T)$ is the utterer of $\hat{p}(T+1)$,
- C is a conjunction of literals in the language of the knowledge base of x. [1]

Any variables in a dialogue constraint are implicitly universally quantified from the outside. The performative $p(T)$ is referred to as the *trigger*, $\hat{p}(T+1)$ as the *next move* and C as the *condition* of the dialogue constraint.

Intuitively, the dialogue constraints of an agent x express policies between x and any other agent. The intuitive meaning of a dialogue constraint $p(T) \wedge C \Rightarrow \hat{p}(T+1)$ of agent x is as follows: if at a certain time T in a dialogue some other agent y utters a performative $p(T)$, then the corresponding instance of the dialogue constraint is triggered

[1] Note that C in general might depend on several time points, possibly but not necessarily including T; therefore we will not indicate explicitly any time variable for it.

and, if the condition C is entailed by the knowledge base of x, then x will utter $\hat{p}(T+1)$, with y as receiver, at the next time $T+1$. This behaviour of dialogue constraints can be achieved by employing an automatic abductive proof procedure such as Fung and Kowalski's IFF [4] within an *observe-think-act* agent cycle [8], as discussed in [14]. The procedure implements a concrete concept of entailment with respect to knowledge bases expressed in abductive logic programming terms [7]. The execution of the proof procedure within the agent cycle allows agents to produce dialogue moves immediately after a dialogue constraint is fired.

The use of an abductive proof procedure, equipped with an operational semantics and theoretical results, has several advantages, beside the availability of an execution model. In particular, the proofs of the theorems stated later in the paper are implicitly grounded on the property of correctness, which holds for the IFF proof-procedure.

An example of a dialogue constraint allowing an agent x to accept a request is:
$tell(Y, x, request(give(R)), T) \ \wedge \ [have(R, T) \wedge not\ need(R, T)]$
$\Rightarrow tell(x, Y, accept(request(give(R))), T+1)$
where the trigger is $tell(Y, x, request(give(R)), T)$, the condition is within square brackets and the next move is $tell(x, Y, accept(request(give(R))), T+1)$.

This is a simple example, but in general C might contain other dialogue performatives, uttered in the past by other agents, within the same dialogue or other dialogues. For instance, one could decide to accept a request only if another agent made a certain move in the past.

We will refer to the set of dialogue constraints associated with an agent $x \in A$ as $\mathcal{S}(x)$, and we will call it the *agent program* of x. We will often omit x if it is clear from the context or unimportant. As already mentioned in Section 2, the agent program of an agent is part of its knowledge base $\mathcal{K}$.

Note that different agents might be equipped with different agent programs.

Definition 5 (*Dialogue*)
A *dialogue* between two agents x and y is a set of ground performatives, $\{p_0, p_1, p_2, \ldots\}$, such that, for some given $t \geq 0$:

1. $\forall\ i \geq 0$, p_i is uttered at time $t+i$;
2. $\forall\ i \geq 0$, if p_i is uttered by agent x (resp. y), then p_{i+1} (if any) is uttered by agent y (resp. x);
3. $\forall\ i > 0$, p_i can be uttered by agent $\alpha \in \{x, y\}$ only if there exists a (grounded) dialogue constraint $p_{i-1} \wedge C \Rightarrow p_i \in \mathcal{S}(\alpha)$ such that $\mathcal{K}(\alpha) \wedge p_{i-1}$ entails C.

A dialogue $\{p_0, p_1, \ldots p_m\}$, $m \geq 0$, is *terminated* if p_m is a ground final move, namely p_m is a ground instance of a performative in $\mathcal{F}(\mathcal{L})$.

By condition 1, a dialogue is in fact a *sequence* of performatives. By condition 2, agents alternate utterances in a dialogue. By condition 3, dialogues are generated by the dialogue constraints, together with the given knowledge base to determine whether the condition of triggered dialogue constraints is entailed.

Intuitively, a dialogue should begin with an initial move, according to the given language for negotiation. The kind of dialogue that is relevant to our purposes is that started with a request for a resource R. Such a dialogue will be initiated by an agent x whose intention $\mathcal{I}$ contains R in its set of *missing* resources.

Definition 6 (*Request dialogue*)
A *request dialogue* with respect to a resource R and an intention $\mathcal{I}$ of agent x is a dialogue $\{p_0, p_1, p_2, \ldots\}$ between x and some agent $y \in A$ such that

– $p_0 = tell(x, y, request(give(R)), t)$,
– $missing(Rs, t) \in \mathcal{I}$ and
– $R \in Rs$,

for some $t \geq 0$.

In general, after a terminated request dialogue d with respect to a resource R and an intention $\mathcal{I}$ of some agent x, the agent might have been successful or not in obtaining the resource R. If it has been successful, then its intention changes so that R is not missing anymore, and the agent will engage in new dialogues to obtain the remaining missing resources, if any, in the intention. If d has been unsuccessful, then the agent's intention does not change, and the agent might decide to engage in a dialogue to obtain R from a different agent. In both cases, the plan in the intention does not change. We also allow a third possibility, of a successful termination with a change of intention possibly (and typically) involving a change of plan. This could happen, for example, if during the dialogue the other agent persuaded the requesting agent to modify its plan, with the promise of a different resource deal. In the sequel, we will assume that *a terminated request dialogue, for a given resource R and intention $\mathcal{I}$, returns an intention $\mathcal{I}'$.*

Definition 7 (*Types of terminated request dialogues*)
Let $\mathcal{I}$ be the intention of some agent a, and R be a missing resource in $\mathcal{I}$. Let d be a terminated resource dialogue with respect to R and $\mathcal{I}$, and $\mathcal{I}'$ be the intention resulting from d. Then, if $missing(Rs), plan(P) \in \mathcal{I}$ and $missing(Rs'), plan(P') \in \mathcal{I}'$:

i) d is *successful* if $P' = P$, $Rs' = Rs \setminus \{R\}$;
ii) d is *conditionally* or *c-successful* if $Rs' \neq Rs$ and $Rs' \neq Rs \setminus \{R\}$;
iii) d is *unsuccessful* if $\mathcal{I}' = \mathcal{I}$.

Note that, in the case of c-successful dialogues, typically, but not always, the agent's plan will change ($P' \neq P$).

In [14], we impose that terminated request dialogues return intentions whose missing resources are no more than those in the original intention, in order to ensure the termination of the negotiation process of an agent, achieved by dialogue sequences. Here, we generalise this property to that of 'convergence':

Definition 8 (*Convergence*)
An agent $x \in A$ is *convergent* iff, for every terminated request dialogue of x, with respect to some resource R and some intention $\mathcal{I}$, the *cost* of the returned intention $\mathcal{I}'$ is not higher than the *cost* of $\mathcal{I}$.

The *cost* of an intention can be defined as the number of missing resources in the intention.

4 Properties of Agent Programs

In this section we consider an individual agent $x \in A$, equipped with a language for negotiation $\mathcal{L}$ and an agent program $\mathcal{S}$ that, for the purposes of this section, we assume to be grounded.

An agent program should ideally guarantee that the agent's dialogues terminate, and that the agent is able to produce one and only one move at a time, in finite time, in response to any non-final move of other agents. In this section, we show how some such desirable properties can be ensured. We discuss the 'termination' property in an earlier paper [14] and in a companion paper [18].

Given a *grounded* agent program $\mathcal{S}$ and a *ground* performative p, we define the set $\mathcal{S}(p)$ of *conditions* associated with p:

$\mathcal{S}(p) \overset{def}{=} \{C \mid \text{ there exists } (p \wedge C \Rightarrow \hat{p}) \in \mathcal{S}\}.$

Definition 9 (*Determinism*)
An agent $x \in A$ is *deterministic* iff, for each performative $p(t)$ which is a ground instance of a schema in $\mathcal{L}^{in}$, there exists at most one performative $\hat{p}(t+1)$ which is a ground instance of a schema in $\mathcal{L}^{out}$ such that $p(t) \wedge C \Rightarrow \hat{p}(t+1) \in \mathcal{S}$ and $\mathcal{K} \wedge p(t)$ entails C.

Namely, an agent is deterministic if for every given performative, at most one of the dialogue constraints triggered by it actually fires, in that it produces a next move.

Determinism can be guaranteed for "non-overlapping" agent programs:

Definition 10 (*Non-overlapping agent program*)
$\mathcal{S}$ is *non-overlapping* iff for each performative p which is a ground instance of a schema in $\mathcal{L}^{in}$, for each $C, C' \in \mathcal{S}(p)$ such that $C \neq C'$, then $C \wedge C' \equiv false$.

Namely, an agent program is non-overlapping if, for any given incoming dialogue move, there do not exist any two triggered dialogue constraints firing at the same time.

Theorem 1 If the (grounded) agent program of x is non-overlapping, then x is deterministic.

Proof (Sketch): for each incoming performative there exists at most one triggered dialogue constraint whose conditions are true, thus $\mathcal{K}$ only entails at most the condition of one constraint. ■

In order for a dialogue not to end before a final move is reached, it is important that an agent be “exhaustive”:

Definition 11 (*Exhaustiveness*)
An agent $x \in A$ is *exhaustive* iff, for each performative $p(t)$ which is a ground instance of a schema in $\mathcal{L}^{in} \smallsetminus \mathcal{F}(\mathcal{L})$, there exists at least one performative $\hat{p}(t+1)$ which is a ground instance of a schema in $\mathcal{L}^{out}$ such that $p(t) \wedge C \Rightarrow \hat{p}(t+1) \in \mathcal{S}$ and $\mathcal{K} \wedge p(t)$ entails C.

Namely, an agent is exhaustive if, for every given performative, at least one of the dialogue constraints triggered by it actually fires, in that it produces a next move.

Exhaustiveness can be guaranteed for “covering” agent programs:

Definition 12 (*Covering agent program*)
Let $\mathcal{L}(\mathcal{S})$ be the set of all (not necessarily ground) performatives $p(T)$ that are triggers in dialogue constraints:
$\mathcal{L}(\mathcal{S}) \overset{def}{=} \{p(T) \mid \text{ there exists } p(T) \ \wedge \ C \Rightarrow \hat{p}(T+1) \in \mathcal{S}\}$. (Obviously, $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}^{in}$)
Then, $\mathcal{S} \neq \emptyset$ is *covering* iff for every performative p which is a ground instance of a schema in $\mathcal{L}^{in}$, $\bigvee_{C \in \mathcal{S}(p)} C \equiv true$ and $\mathcal{L}(\mathcal{S}) = \mathcal{L}^{in} \smallsetminus \mathcal{F}(\mathcal{L})$.

Namely, an agent program is covering if, for any given incoming dialogue move that is not a final move, there exists at least one triggered dialogue constraint which actually fires.

Theorem 2 If the (grounded) agent program of x is covering, then x is exhaustive.
Proof (Sketch): for each incoming performative there exists at least one dialogue constraint which is triggered and whose condition is true, thus $\mathcal{K}$ entails the condition of at least one constraint. ■

If an agent is exhaustive and deterministic, then at each step of the dialogue the agent will produce *exactly one* reply to a (non final) move made by the other agent. If both agents involved in a dialogue are exhaustive and deterministic, then at each step exactly one agent is guaranteed to produce only one dialogue move, unless a final move is made. [2] The IFF proof procedure and the agent cycle have features that allow us to produce dialogues as defined earlier, given agent programs that are exhaustive and deterministic.

5 Agent Varieties: Concrete Examples of Agent Programs

In this section we show some concrete agent programs and we discuss which properties hold for them and which ones do not. All the examples refer to a given agent a, equipped with a language for negotiation $\mathcal{L}_2 = \mathcal{I}(\mathcal{L}_2) \cup \mathcal{F}(\mathcal{L}_2)$, where:

- $\mathcal{I}(\mathcal{L}_2) = \{\ tell(X, Y, \mathbf{request}(\mathbf{give}(Resource)), T)\ \}$;
- $\mathcal{F}(\mathcal{L}_2) = \{\ tell(X, Y, \mathbf{accept}(Move), T), tell(X, Y, \mathbf{refuse}(Move), T)\ \}$.

[2] Note that we use *transaction*, rather than *real*, time, i.e., we assume that there is no time-point between two *consecutive* dialogue moves in a dialogue.

Agent program `P0`

$tell(X, a, \mathbf{request}(\mathbf{give}(R)), T) \ \wedge \ have(R, T)$
$\Rightarrow tell(a, X, \mathbf{accept}(\mathbf{request}(\mathbf{give}(R))), T+1)$
$tell(X, a, \mathbf{request}(\mathbf{give}(R)), T) \ \wedge \ need(R, T)$
$\Rightarrow tell(a, X, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(R))), T+1)$

This program is neither deterministic nor exhaustive. In fact, if a has and needs the resource that is requested at time T, the program allows to return two answers at time $T+1$, namely both *accept* and $refuse$, while if a does not have R, then the program will not be able to return any answer at time $T+1$.

Agent program `P1`

$tell(X, a, \mathbf{request}(\mathbf{give}(R)), T) \ \wedge \ have(R, T)$
$\Rightarrow tell(a, X, \mathbf{accept}(\mathbf{request}(\mathbf{give}(R))), T+1)$
$tell(X, a, \mathbf{request}(\mathbf{give}(R)), T) \ \wedge \ not\ have(R, T)$
$\Rightarrow tell(a, X, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(R))), T+1)$

An agent programmed with `P1` simply gives away the requested resource if it has it, no matter whether it needs it or not. This program is exhaustive and deterministic.

Agent program `P2`

$tell(X, a, \mathbf{request}(\mathbf{give}(R)), T) \ \wedge \ have(R, T) \ \wedge \ not\ need(R, T)$
$\Rightarrow tell(a, X, \mathbf{accept}(\mathbf{request}(\mathbf{give}(R))), T+1)$
$tell(X, a, \mathbf{request}(\mathbf{give}(R)), T) \ \wedge \ not\ have(R, T)$
$\Rightarrow tell(a, X, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(R))), T+1)$
$tell(X, a, \mathbf{request}(\mathbf{give}(R)), T) \ \wedge \ need(R, T)$
$\Rightarrow tell(a, X, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(R))), T+1)$

An agent programmed with `P2` gives away the requested resource only if it has it and it does not need it. This program is exhaustive and deterministic.

6 Dialogue Sequences

In this section we model the negotiation process of an agent by means of "sequences of (request) dialogues" with respect to an intention, I, of a given agent x. The request dialogues aim at obtaining *all* the missing resources in I. We assume that all dialogues between x and any other agents are *atomic*, i.e., it is not possible for x to be engaged in two different dialogues at the same time.

In order to start a request dialogue with respect to an intention I, the set of missing resources of I must be non-empty, i.e., $missing(Rs) \in I, Rs \neq \emptyset$.

Intuitively, given a goal G such that $goal(G) \in I, missing(Rs) \in I$ and $plan(p) \in I$, a sequence of dialogues with respect to I aims at decreasing the cost of I, either by obtaining one by one all the missing resources and making p executable, or by finding an alternative plan (and thus an alternative intention) which can be made executable. Indeed, after each dialogue, the intention might change. We associate with a dialogue sequence $d_1, d_2, \ldots, d_n$ the corresponding sequence of intentions $I_1 = I, I_2, \ldots, I_{n+1}$. In general, after a request dialogue $d_i, i < n$, with respect to an intention I_i and a resource R, is terminated, the agent will have an intention I_{i+1}. All such intentions have the same goal, but possibly different plans and missing/available resources.

Definition 13 (*Sequence of dialogues*)
A *sequence of dialogues* $s(I)$ with respect to an intention I of an agent x with $goal(G) \in I$ is an ordered set $\{d_1, d_2, \ldots, d_n, \ldots\}$, associated with a sequence of intentions I_1, I_2, ..., $I_{n+1}, \ldots$ such that

- for all $1 \leq i, d_i$ is a request dialogue between x and some other agent (not necessarily the same for all i), with respect to a resource R_i such that $R_i \in Rs_i, missing(Rs_i) \in I_i, I_i$ is the intention of agent x, $goal(G) \in I_i$, and I_{i+1} is the intention after dialogue d_i;
- $I_1 = I$;

We are interested in dialogue sequences that terminate:

Definition 14 (*Termination of a dialogue sequence*)
A sequence of dialogues $\{d_1, d_2, \ldots, d_n\}$ with respect to an initial intention I of an agent x and associated with the sequence of intentions $I_1, I_2, \ldots, I_{n+1}$ is *terminated* iff there exists no possible request dialogue with respect to I_{n+1} that x can start.

Termination could occur for two reasons: either there are no missing resources in I_{n+1} or x could not obtain all the resources in I_{n+1} but "decided" to give up the intention without forming a new one for the time being. Accordingly, we can assess the success of dialogue sequences:

Definition 15 (*Success of a dialogue sequence*)
A terminated sequence of dialogues $\{d_1, d_2, \ldots, d_n\}$ with respect to an initial intention I of an agent x and associated with a sequence of intentions $I_1, I_2, \ldots, I_{n+1}$ is *successful* if I_{n+1} has an empty set of missing resources; it is *unsuccessful* otherwise.

In the remainder of this section we will discuss how dialogue sequences can be generated by agents, by an "agent dialogue cycle" which repeatedly generates individual dialogues until a terminated sequence is produced. We will call SD and SI, respectively, the sequence of dialogues and intentions, produced within the dialogue cycle:

Definition 16 (*Agent dialogue cycle*)
Given an initial intention I of agent x, containing a set of missing resources Rs, the *agent dialogue cycle* is the following:

1. set $SD = \emptyset$ and SI to $\{I\}$;
2. until $Rs = \emptyset$ repeat
   (a) select a resource $R \in Rs$ and update Rs to $Rs - \{R\}$;
   (b) until a successful or c-successful dialogue d is found, between x and some other agent, repeat

```
            i. select an agent $y \in A - \{x\}$ such that there does not exist in $SD$ a request
               dialogue between $x$ and $y$ with respect to $R$;
           ii. if such $y$ does not exist, then **failure** and restore $Rs$ to $Rs \cup \{R\}$;
          iii. construct a request dialogue $d$ between $x$ and $y$ with respect to $R$ and $I$,
               returning an intention $I'$;
       (c) add $d$ to $SD$ and $I'$ to $SI$;
       (d) update $I$ to $I'$ and $Rs$ to $Rs'$ such that $missing(Rs') \in I'$;
3. **success**.
```

Note that the above agent dialogue cycle does not allow for the plan in an intention to change unless the change is achieved by an individual dialogue between x and the chosen y.

Note also that the above agent dialogue cycle has the following properties:

– no agent is asked twice for the same resource within the dialogue sequence;
– if a resource is not obtained from one agent, then it is asked from some other agent, if any;
– if a resource is not obtained after asking all agents, then the agent dialogue cycle terminates with **failure**.

The idea behind the third property is that the agent will not carry on asking for the other resources, since, as at least one resource in the current intention cannot be obtained, the intention will not possibly be executable.

Theorem 3 Given an agent $x \in A$, if x's agent dialogue cycle returns **success** then there exists a successful dialogue sequence with respect to the initial intention I of x.

Proof (Sketch): The dialogue sequence SD, associated with the sequence of intentions SI, as generated by the agent dialogue cycle at step 2(c) during the last execution of the loop at step 2, is a successful dialogue sequence. ■

If the agent $x \in A$ is convergent, then all dialogues of a sequence with respect to an intention I of x will result in a non-increased cost of x's intention. Thus, we can set an upper bound to the number of dialogues in a sequence initiated by x:

Theorem 4 Given an agent x with intention I, and a successful dialogue sequence $s(I)$ generated by x's dialogue cycle, if x is convergent, then the number of dialogues in $s(I)$ is bounded by $m \cdot |Rs|$, where $missing(Rs) \in I$ and $|A \setminus \{x\}| = m$, A being the set of agents in the system.

Proof (Sketch): $|Rs|$ can never increase, since x is convergent. In the worst case, x will have to request each resource $R \in Rs$ to all the other m agents of the system, thus generating m dialogues with respect to each resource. The cardinality of the set of missing resources decreases by only one unit after each successful dialogue. ■

7 Using Dialogue Sequences for Resource Reallocation

In this section, we study the use of multiple sequences of dialogues for the solution of the well-known and widely studied *resource reallocation problem* [2,16]. The problem can be rephrased in our system as follows.

Definition 17 (*Resource reallocation problem – r.r.p.*)
Given an agent system A, with each agent $x \in A$ equipped with a knowledge base $\mathcal{K}(x)$ and an intention $\mathcal{I}(x)$,

- the *r.r.p. for an agent* $x \in A$ is the problem of finding a knowledge base $\mathcal{K}'(x)$, and an intention $\mathcal{I}'(x)$ (for the same goal as $\mathcal{I}(x)$) such that $missing(\emptyset) \in \mathcal{I}'(x)$.
- the *r.r.p. for the agent system* A is the problem of solving the r.r.p. for every agent in A.

A *r.r.p.* is *solved* if the required (sets of) knowledge base(s) and intention(s) is (are) found.

The new knowledge bases will differ from the old ones in the resources available to the agents and in the intentions. Typically, agents need to acquire new resources from other agents so that their intentions have no missing resources. In our framework, agents can obtain the new resources by means of dialogue sequences.

Note that, the knowledge and intention of an agent x might change not only as a result of dialogues which have been initiated by x, but also as a result of dialogues initiated by other agents, but engaging x. Indeed, in such more 'passive' dialogues, x might agree to give away a resource that it needs. This 'double change' of knowledge and intentions (of both agents involved in a dialogue) plays a role when agents need to engage in multiple concurrent dialogue sequences in order to achieve their goal. [3] In the remainder of this section, we will assume *generalised notions of (terminated) dialogue*, modifying the intentions of both agents engaged in it, and *dialogue sequence*, modifying the intentions of all agents involved in all dialogues in it. Then, the notion of *convergence* can be generalised as well, so that the cost of both agents' intentions returned by a dialogue is not increased, with respect to the cost of the agents' initial intentions. Note that convergent agents, in this generalised sense, correspond to *self-interested individual rational* agents as proposed in the literature (see Sandholm [15]).

We consider two properties of agents:

- *correctness*: if the dialogue cycle of some agent x returns success, then the r.r.p of x is solved, and, if the dialogue cycle of every agent in A returns success, then the r.r.p. of the agent system is solved;
- *completeness*: if there exists a solution to the r.r.p of an agent system, then the dialogue cycle of every agent in A returns success, producing such a solution.

The proof of correctness for a single agent is straightforward, by definition of successful dialogue sequence. In the case of an agent system, instead, correctness as given above can be proven only for convergent agents.

Theorem 5 (*Correctness of the agent dialogue cycle with respect to the r.r.p.*) Let A be the agent system, with the agent programs of all the agents in A being convergent. If

[3] Note that this is not in contrast with the assumption that agents engage in at most one dialogue at a time. As we assume that dialogues are atomic, there will always be at most one dialogue taking place at any time.

all agent dialogue cycles of all agents in A return **success** then the r.r.p. for the agent system is solved.
Proof (Sketch): Since all agent programs are convergent, there will never occur a dialogue after which, and as a consequence of which, either of the two agents involved has more missing resources in its intention than it had before. Therefore, having m successful dialogue sequences, one for each agent in A, is in the worst case equivalent to having m *independent* dialogue sequences, after each of which the relevant agent will have solved its r.r.p., thus solving the r.r.p. of A. ■

In order to address completeness we need to deal with the issue of how intentions change, which is beyond the scope of this paper. But to illustrate the problem, consider the following example. Let $A = \{a, b, c\}$ be an agent system, with knowledge bases containing:
$a : \mathcal{R}(a) = \emptyset, \mathcal{I}(a) = \{ \mathtt{plan}(\{\mathtt{p(a)}\}), \mathtt{missing}(\{\mathtt{r_2}\}), \mathtt{available}(\emptyset), \mathtt{goal}(\{\mathtt{g(a)}\})\}$
$b : \mathcal{R}(b) = \{r_2\}, \mathcal{I}(b) = \{ \mathtt{plan}(\{\mathtt{p(b)}\}), \mathtt{missing}(\{\mathtt{r_3}\}), \mathtt{available}(\{\mathtt{r_2}\}), \mathtt{goal}(\{\mathtt{g(b)}\})\}$
$c : \mathcal{R}(c) = \{r_3, r_4\}, \mathcal{I}(c) = \{ \mathtt{plan}(\{\mathtt{p(c)}\}), \mathtt{missing}(\emptyset), \mathtt{available}(\{\mathtt{r_3}\}), \mathtt{goal}(\{\mathtt{g(c)}\})\}$
where $\mathcal{R}(x)$ stands for the resources available to agent x, as described within x's knowledge base. Let us assume that there exists an alternative plan $\mathtt{p'(b)}$ for b's goal $\mathtt{g(b)}$, and therefore an alternative intention, $\mathcal{I}'(b)$: $\mathcal{I}'(b) = \{ \mathtt{plan}(\{\mathtt{p'(b)}\}), \mathtt{missing}(\{\mathtt{r_4}\}), \mathtt{available}(\emptyset), \mathtt{goal}(\{\mathtt{g(b)}\})\}$ It is easy to see that there exists a possible resource reallocation that makes all goals achievable (r_2 to a, r_4 to b and r_3 to c), but in order to be found by negotiation, b must be able to change its intention to $\mathcal{I}'(b)$.

We consider a weak notion of completeness, whereby agents have, as their own initial intentions, the intentions whose plan does not need to be changed by negotiation.

Definition 18 (*Weak completeness*)
Let A be an agent system consisting of n agents. Let $\mathcal{R}(A)$ be the union of all resources held by all agents in A, and $\mathcal{R}(\mathcal{I}(A))$ be the union of all resources needed to make all agents' initial intentions $\mathcal{I}(A)$ executable.

A is *weakly complete* if, given that $\mathcal{R}(\mathcal{I}(A)) \subseteq \mathcal{R}(A)$, then **there exist** n successful dialogue sequences, one for each agent in A, such that the intentions $\mathcal{I}'(A)$ returned by the sequences have the same plans as $\mathcal{I}(A)$ and all have an empty set of missing resources.

Theorem 6 (*weak completeness with respect to* `P1 + P2`) Let A be an agent system consisting of n agents. Then, if the agent program of every agent in A is either `P1` or `P2`, then A is weakly complete.
Proof (Sketch): The sub-system composed by the agents programmed with `P2` is weakly complete, therefore the agents programmed with `P2` will obtain all missing resources, either from the agents programmed with `P1` or from the ones programmed with `P2`. The agents programmed with `P1` will obtain the missing resources either from the agents programmed with `P2` that have such resources and do not need them, or by the agents programmed with `P1`, which represent also a weakly complete sub-system. ■
Thus, we have shown that the r.r.p. can be solved within our approach, by diverse agents, equipped with different agent programs, negotiating with one another. Note that such

diversity can be used to implement a concept of priority between agents. Indeed, agents programmed with P2 always retain all resources they need, whereas agents programmed with P1 give them away, if asked. Thus, P2-agents have a higher priority than P1-agents, with respect to the allocation of resources.

8 Conclusions

This paper focused on one-to-one agent negotiation for achieving goals in a system with limited resources. The solution proposed is based on agent dialogues, as a way of requesting resources, proposing resource exchanges, and suggesting alternative resources. The negotiation framework presented is a general one for representing policies and generating dialogues. The resource reallocation problem is one possible application of this framework. This framework could be deployed to tackle other problems, e.g. to deal with dialogues starting with an offer or an advertisement rather than a request.

We assume that the agents are provided with a planner that, given a goal, is able to produce a sequence of actions to achieve it and an associated set of missing resources that are needed to carry out the actions. The framework proposed is independent of the planner, the specific application domain, the agent programs that regulate the dialogues, and even the language that agents use to communicate.

In the domain of resource reallocation, a single dialogue is used to obtain a resource. Our approach is logic based, and, in particular, the agents involved in a dialogue can use an abductive proof procedure to reason about their current beliefs and intentions, and produce a dialogue move. An agent dialogue cycle is used produce sequences of dialogues, and in general sets of sequences of dialogues in order for the agent to collect all the resources it needs to achieve its goal.

The logic-based approach allows us to define and prove some interesting properties of dialogues, sequences of dialogues, and sets of sequences of dialogues in an agent system. In particular, the adoption of an abductive proof procedure for which termination, correctness and completeness results hold, allows us to extend such general results to the specific case of a problem defined and addressed in the paper, the resource reallocation problem. We show various agent programs that can be used to tackle such a problem, and discuss to what extent the properties of termination, correctness and completeness, as defined in this paper, hold for them.

Introduction and surveys of automated negotiation can be found in [13] and [9,6], respectively. An approach to negotiation via dialogue, that makes use of an argumentation system, is that of Sycara [17], and more recently Kraus et al. [10], where the authors adopt a modal logic approach and focus on the mental states of agents that move towards an agreement and encourage cooperation. The authors' formalisation of dialogues makes use of rules that, similarly to our dialogue constraints, are triggered when a performative is received, and may lead to another dialogue move under certain conditions, within an agent cycle. The execution is intertwined with the dialogue, allowing the agents to express threats, promises of rewards, etc. and to maintain a mental model of the other partners. Such an approach, focused on persuasion, leads to different results from ours, being somewhat more descriptive, and not aiming to determine specific properties of the negotiation policy.

Argumentation-based approaches to agent negotiation, that are in a way more general and less operational than ours, but are still very relevant, are those of Amgoud et al. [1] and Parsons et al. [11], from which we inherited the nail-and-hammer example.

Zlotkin and Rosenschein [19] adopt a game theory approach to the study of automated negotiation. However, their work – as most others that are game theory-based – considers agents that are not cooperative, and focuses on conflict resolution rather than collaboration. One of the interesting issues they address is the disclosure of private information. Our work is somewhat orthogonal, aiming at obtaining the resources the agents need, without taking into account the problems that could arise from disclosing private information. We agree that this is very important, especially in contexts such as e-commerce and virtual enterprises.

The use of negotiation to tackle the resource reallocation problem is not a new idea; among other contributions we cite that by Faratin [3], Sandholm [15], and Sathi and Fox [16]. Most of them study the problem adopting a game-theoretic approach and prove results that we consider orthogonal to ours.

This paper contributes to a number of issues. The introduction of a logic-based framework for negotiation allows us to prove properties that we can use to forecast the behaviour of a system of agents with no need for simulation and experimental results. The paper suggests an execution model that relies on the *observe-think-act* agent cycle of Kowalski and Sadri [8], and on the adoption of the IFF abductive proof procedure of Fung and Kowalski [4]. In this way, the implementation of the system, at least as far as a prototype, is straightforward, and does not require much more than writing high-level dialogue rules of the kind presented in the paper. This work extends our previous work [14] by dealing with dialogue sequences, discussing the application of the framework to a general resource reallocation problem, and proposing varieties of agents that can tackle such a problem.

There are a number of open issues. So far we assumed that agents do not change their goals. A natural extension would be to take into account the possibility of modifying goals as a consequence of an unsuccessful dialogue sequence, in accordance with the BDI theory that requires goals to be believed feasible.

Negotiation of sets of resources, rather that single items, is another interesting problem, as is the use of broadcast or multi-cast primitives as in electronic auctions.

It would also be interesting to study different cost functions for intentions, and to see what their impact is on the dialogues and on the dialogue properties. Currently, in moving from one intention to another, as a consequence of a deal (the acceptance of a $promise$), an agent can choose a plan that turns out not to be feasible, because of the lack of resources in the system. A more sophisticated cost function than the one proposed in the paper could result in the agent ensuring that all the resources for a new plan exist somewhere in the system, before agreeing to a deal.

Acknowledgements

Thanks to Kostas Stathis and Paolo Mancarella for helpful discussions, and to the anonymous referees for useful comments.

References

1. L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue and negotiation. In W. Horn, editor, *Proceedings 14th European Conference on Artificial Intelligence, Berlin, Germany*. IOS Press, August 2000.
2. M. Barbuceanu and W. Lo. A multi-attribute utility theoretic negotiation architecture for electronic commerce. *Proceedings of the Fourth International Conference on Autonomous Agents, Barcelona, Spain*, pages 239–247, 2000.
3. P. Faratin. *Automated Service Negotiation Between Autonomous Computational Agents*. PhD thesis, Department of Electronic Engineering, Queen Mary College, University of London, UK, 2000.
4. T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 1997.
5. N. R. Jennings. Automated haggling: Building artificial negotiators (invited talk). In *AISB'01 Convention, York, UK*, March 2001.
6. N. R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, C. Sierra, and M. J. Wooldridge. Automated Negotiation: Prospects, Methods and Challenges. *International Journal of Group Decision and Negotiation*, 10(2), 2001.
7. A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. *Handbook of Logic in AI and Logic Programming*, 5:235–324, 1998.
8. R. A. Kowalski and F. Sadri. From logic programming to multi-agent systems. *Annals of Mathematics and AI*, 1999.
9. S. Kraus. *Strategic Negotiation in Multi-Agent Environments*. MIT Press, Cambridge, MA, 2000.
10. S. Kraus, K. Sycara, and A. Evenchik. Reaching agreements through argumentation; a logical model and implementation. *Artificial Intelligence*, 104:1–69, 1998.
11. S. Parsons, C. Sierra, and N. R. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.
12. A. Rao and M. Georgeff. An abstract architecture for rational agents. In *Proceedings of the International Workshop on Knowledge Representation (KR'92)*, 1992.
13. J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, Cambridge, Massachusetts, 1994.
14. F. Sadri, F. Toni, and P. Torroni. Logic agents, dialogues and negotiation: an abductive approach. In *Proceedings AISB'01 Convention, York, UK*, March 2001.
15. T. Sandholm. *Negotiation among Self-Interested Computationally Limited Agents*. Computer science, University of Massachusetts at Amherst, September 1996.
16. A. Sathi and M. Fox. Constraint-directed negotiation of resource reallocation. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence*, volume II, pages 163–195. Morgan Kaufmann, San Mateo, CA, 1989.
17. K. P. Sycara. Argumentation: Planning other agents' plans. In *Proceedings 11th International Joint-Conference on Artificial Intelligence*, pages 517–523. Morgan Kaufman, 1989.
18. P. Torroni. A study on the termination of negotiation dialogues. Technical Report DEIS-LIA-01-008, University of Bologna (Italy), 2001. LIA Series no. 52.
19. G. Zlotkin and J. S. Rosenschein. Mechanisms for automated negotiation in state oriented domains. *Journal of Artificial Intelligence Research*, 5:163–238, 1996.

Agents for Hand-Held, Mobile, or Embedded Devices

Tim Finin

U. of Maryland Baltimore County
finin@cs.umbc.edu

This section consists of papers presented in a special session at ATAL 2001, devoted to a challenging new area of agent technology, viz. agents for hand-held, mobile, or embedded devices, such as cell phones and palmtops. The challenge in this area is to do agent-based computing with the highest functionality possible on devices with very limited resources. In particular, one likes to have agents running on them with adequate communication capabilities, preferably FIPA-compliant. The papers in this section address the problem of how to obtain these agent communication capabilities in small devices.

KSACI: A Handheld Device Infrastructure for Agents Communication[*]

Ryan L. Albuquerque[1], Jomi F. Hübner[2], Gustavo E. de Paula[1], Jaime S. Sichman[2], and Geber L. Ramalho [1]

[1] Centro de Informática, Universidade Federal de Pernambuco
Cx. Postal 7851
50732-970, Recife, PE, Brasil
{rla2,gep,glr}@cin.ufpe.br
[2] Laboratório de Técnicas Inteligentes
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, 158, tr. 3
05508-900, São Paulo, SP, Brasil
{jomi.hubner,jaime.sichman}@poli.usp.br

Abstract. The recent development of software platforms for cell phones and handheld computers, such as Java 2 Micro Edition (J2ME), has broadened application perspectives in this area. The developers can now write their own software to run in handheld devices, what was impossible recently since the platforms were proprietary. Among the myriad of applications for these devices, some of them are very complex thus requiring the intelligent behavior typically provided by agents. However, since J2ME is a very recent platform, there are no well-established J2ME-based environments or tools for agent development yet. In this context, it is extremely helpful to develop building-block components, such as deductive inference mechanisms and communication languages and protocols. This paper describes the KSACI, a tool that provides communication infrastructure among agents running in handheld devices. KSACI supports KQML, as the outer language, and XML, as the inner one. KSACI extends SACI (Simple Agent Communication Infrastructure), a Java open-source communication infrastructure for desktop agents. Together with two other works presented in this book, KSACI represents a pioneer effort in the development of such communication tools. KSACI is already fully implemented and its preliminary test results on cell phone emulators are encouraging.

1 Introduction

This century is witnessing a new trend in computer science research: the pervasive computing. According to it, computation will be increasingly embedded in small mobile and connected devices, providing to users relevant information and services anytime and anywhere [1]. One recent effort for the actual implementation of

[*] This work is partially supported by CNPq, Brazil, grants 301041/95-4 and 520263/99-5; and by CNPq/NSF PROTEM-CC MAPPEL project, grant 680033/99-8.

computer pervasiveness is the specification and implementation of Java 2 Micro Edition (J2ME, also called KJava) [2;3]. J2ME slims down the Java Standard Edition (J2SE) [4] by removing or rewriting key parts of the core runtime environment in order to fit it into small devices. As an outcome of a consortium involving Sun Microsystems Inc. and the major telecommunication devices manufactures (e.g., Motorola, Nokia, Sony, Samsung and Ericsson), J2ME opens new application perspectives in this area. In fact, it allows the developers to write their own software for handheld devices, which was not possible before, since the platforms for these devices were proprietary.

The J2ME effort integrated with the novel handheld devices' capabilities (such as location, storage, processing and communication) favors a myriad of applications from which users can daily benefit. These applications may range from simple e-mail systems to complex applications, such as intelligent Personal Digital Assistants, interactive multiplayer games, e-commerce location-sensitive transactions systems, and so on. Some of these applications require the intelligent behavior typically provided by agents, since they must exhibit capabilities such as autonomy, goal-driven reasoning, reactivity, adaptation, as well as coordination and cooperation with other software entities.

Let us consider, for instance, the case of a shopping center in which each shop may have an agent that uses its "spatial tracking" capabilities to identify whether passing by users (carrying a handheld device with an embedded agent) are regular clients of that shop. If this is the case, the shop agent can suggest some novel products that may interest the user. The device agent reacts by evaluating the suggestion with respect to its latest model of the owner's interest profile stored in the device. If the shop agent suggestion matches that profile, the two agents may then interact to negotiate price and payment conditions. In another scenario, a user may ask his/her agent to book two tickets for the cinema tonight. Based on the user's preferences regarding movie genders, schedules and cinema, the agent could remotely check the guest's agenda about the available time and contact some entertainment brokering agents in order to choose the configuration which best fits the user's requirements. After booking the appointment, this agent would authorize the payment and then notify the user and his/her guest. When the user arrives at the cinema's entrance, the handheld device agent is triggered to ask the movie house agent to open the roulette for the two paid tickets.

In order to embed agents in J2ME-based handheld devices, it would be extremely helpful to reuse Application Programming Interfaces (APIs) devoted to implement some building-blocks components for agents, such as deductive inference mechanisms and communication languages and protocols. Unfortunately, since J2ME has only recently been released, there are no well-established J2ME-based environments or tools development.

The problem of providing inference mechanisms is perhaps not critical at the moment, since several applications can be developed using reactive agents alone or coupled with a PC/Workstation server, working as a cognitive agent partner. There are already some J2SE-based tools to support agent's deductive inference, such as JEOPS [5;6], JESS [7] InterProlog [8] and JavaLog [9]. For this reason, our project has been targeted to solve firstly the problem of providing agents communication infrastructure for handheld devices.

This paper describes KSACI, which is, together with two other works presented in this book [15;16], a pioneer effort on providing agent communication infrastructure. KSACI extends SACI (Simple Agent Communication Infrastructure) [10;11], a Java open-source agent communication infrastructure, in order to enable handheld devices embedded agents to exchange information and knowledge with other embedded agents or with agents located in desktop computers (PCs and Workstations). This tool works with KQML [13;14], as the an agent communication protocol, as well as with XML and text as content language. KSACI is already fully implemented and its preliminary test results, on cell phone emulators, are encouraging.

In Section 2, we enumerate some requirements for the development of an agent communication component for handheld devices. In Section 3, we explain the main concepts of J2ME, showing that, despite some limitations, it can be used to implement this communication component. Then, we present the KSACI architecture and its relationship with other technologies (XML, KQML and SACI). Finally, we draw some conclusions and discuss future research directions.

2 Requirements

The development of agent-based applications for hand-held devices, it would be helpful to reuse APIs such as those dedicated to reasoning and communication. In order to develop a communication API, which is the focus of this paper, one should pay attention to some requirements regarding three important aspects: the programming language, the agent communication language and the communication infrastructure.

2.1 Programming Language

A programming language that supports the development of device-targeted applications is necessary. This language should be slim enough to meet the severe restrictions of memory and processing power. It should also be a platform independent language, so that applications can be developed with high compatibility among different device platforms like PDAs, cell phones and two-way pagers. In addition, it should make available all object oriented software engineering modeling characteristics like reusability and modularity.

2.2 Agent Communication Language

In a Multi-Agent System (MAS), the agents solve problems collectively through coordination, cooperation and/or negotiation [15;18]. Since they do not necessarily run on the same platform or have the same knowledge representation model, they need to share a common ontology and a common communication protocol, so that they can communicate effectively exchanging information and knowledge. The use of an Agent Communication Language (ACL) is particularly required in a highly heterogeneous environment, which is the case of handheld devices. ACL can provide high-level communication interoperability among these devices.

ACL is composed of an *outer* and an *inner language* [14]. An outer language is independent of platform, content, ontology and network transport protocol and is responsible for encapsulate the main attributes of messages in the communicative act. The language where the content of a message is written is commonly called an *inner language* (e.g. KIF, Prolog, Lisp, XML and Java). The inner language should be structured enough to facilitate parsing and expressive enough to represent the required knowledge.

2.3 Communication Infrastructure

The communication infrastructure is responsible for enabling agents to communicate through a physical network (wireless or not) hiding the complexity of its protocols.

In a handheld device environment, the agents are distributed into a wireless network and will have to interact with each other. The implementation of multi-agent systems in such environment is sometimes difficult, since the project designer should to be aware, besides the agent's intelligent features, of distributed programming technologies like RMI [19], CORBA [20], DCOM [21], etc.

Concerning the variety of devices, it will be interesting to rely on a platform-independent infrastructure that hides these complexities using a communication protocol. Besides all these characteristics, the network standard requirements, such as reusability, modularity and efficiency, should be considered.

3 Java 2 Platform, Micro Edition (J2ME)

Before the release of J2ME [2], the structure of the handheld device computing industry was essentially as fragmented and vertical as was the conventional computing industry back in the 60s: a wide variety of hardware vendors, each one also being the sole provider of infra-structure software (operating systems, network protocols) running on their hardware, and the sole provider of application software running on top of such infra-structure software. Third party development of locally running applications was limited to some high-end handheld devices, palmtops and PDAs, for which C compilers had been released by the hardware vendors. Even for such platforms, the lack of standardization drastically limited the portability of third party applications.

The release of J2ME made possible third party development of applications that need to be written only once and can then run on the whole spectrum of computing devices, provided that there is a J2ME virtual machine available for them. Such virtual machine has been embedded in the most recent handheld devices produced by the many key vendors that have endorsed J2ME: Motorola, Siemens, Nokia, Ericsson, Samsung, etc.

J2ME (also called KJava) has slimmed-down the Standard Edition by removing or rewriting key parts of the core runtime environment to be able to fit in small devices. It is based on a three-layer software architecture sitting on top of the device's operating system, as shown in Figure 1.

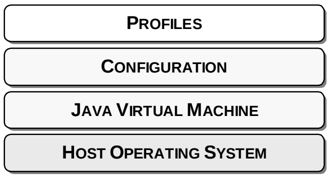


Fig 1. Three-Layered Model of J2ME

The *Java Virtual Machine Layer* is an implementation of the core Java virtual machine that is customized for both a particular underlying operating system and a particular overlying J2ME configuration. The *Configuration Layer* extends the core language virtual machine with a minimum class library provided by J2ME *for a particular category of devices associated with a particular range of resource limitations*, such as medium devices (net TV, set-top boxes, or network PCs) or small devices (pagers, cell-phones or smart-phones). *The Profile Layer* implements an extended Application Programming Interface (API) that in turn extends the configuration class library with additional classes encapsulating common services needed *for a particular category of application domains*.

This three-layer architecture acknowledges the practical impossibility to provide a fully uniform set of services across hardware platforms as diverse as smart cards and network PCs, while still allowing for a high-level of software reuse. It does so by judiciously separating three largely orthogonal hurdles to code reuse and portability: (1) different operating systems, (2) different resource limitations, (3) different types of built-in services required from the underlying platform by the applications.

So far, there are two configurations already developed and made available:

- CDC (Connected Device Configuration): targeted to shared, fixed and connected information devices like TV set boxes, Internet TVs, Internet-enabled screen-phones, high-end communicators, and automobile entertainment / navigation systems;
- CLDC (Connected Limited Device Configuration) [22;23]: targeted to personal, mobile and connected information devices such as cell phones, pagers and personal organizers.

The only profile developed so far was built on top of the CLDC: the MIDP (*Mobile Information Device Profile*) [24;25]. There are other profiles under development such as *PDA* (built on top of CLDC), *Foundation Profile* (built on top of CDC), *Personal Profile* (add-on profile built on top of Foundation Profile) and *RMI Profile* (add-on profile built on top of Foundation Profile) [3] (see Figure 2).

As mentioned earlier, J2ME is a slimed-down version of J2SE, from which a set of built-in functionalities has been taken out [3]. The three key functionalities that are absent in J2ME are *dynamic loading of user-defined classes*, *reflection* and *floating point*.

For security reasons, an application cannot influence in any way in which classes are loaded. Thus, the only class loader available to applications running on the KVM is the one provided by the KVM itself. This means that an application written in J2ME cannot locate and load classes other than those from core runtime and the application own classes.

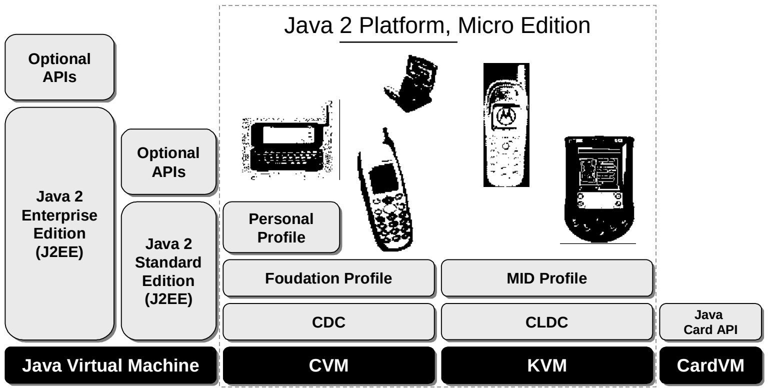


Fig. 2. Java 2 Platforms

The absence of reflection eliminates some J2SE important features, such as object serialization. Consequently, the agents will not be able to use Remote Method Invocation (RMI) [19] to transport their messages. All network communication is done via sockets.

Since the floating-point operations are especially expensive without a dedicated coprocessor, the KVM is not able to support any floating-point types or constants and nor any byte code involving floating point operations.

When developing a mobile application, one can run into problems in dealing with the above J2ME limitations. For instance, in agent communication, some message contents may be misunderstood if they refer to floating points values (e.g. in e-commerce transactions). The seemingly obvious technique is to take all missing J2SE classes and add to the J2ME application. However, this approach cannot be always used, since some classes may have native methods.

The deployment of a J2ME application is done via two files: a Java archive (JAR file) and a Java application descriptor (JAD file). The JAR file is a compressed file with all the application pre-verified classes. This process is needed in J2ME mainly due to security and performance reasons. The JAD file is an application descriptor which will inform the application name, size, packages, jar file location, and other relevant information needed in order to the application be downloaded. Most of this information is also present in the standard manifest file, into the JAR file, to enable the application to run.

4 KSACI

This section describes how we have dealt with the J2ME restrictions previously discuss, in order to provide means for agents in handheld devices to communicate with other agents. In the first part we present a J2SE tool called SACI [10;11] that served as the agent communication infrastructure. After that, the KSACI architecture, which extends SACI, is presented, and a solution for the problem of the messages' contents is described.

4.1 SACI

SACI is a communication infrastructure developed by our research team as an Java open source API, which can easily modified by extending Java classes. It attends to the requirements presented earlier in section 2.3. SACI works currently with the KQML (the support to FIPA ACL is being implemented), support white and yellow page service and is extremely efficient as discussed later. Because of these good characteristics and the fact that we have been working with SACI for a while, we decided to extend it to J2ME, instead of adapting analogous tools (e.g. Jackal, FIPA OS and JKQML).

SACI has a society model based on client-server platform, where a society is composed of agents, one of them called facilitator or router. This agent is responsible to provide all the services of a society like white and yellow pages, registering and announcing skills. The internal behavior of SACI agents is simple. The agent has to *enter* into a society; then, *exchange* messages and *announce* skills; and at last, *exit* the society. Each society has one (and only one) facilitator to help messages delivery. The facilitator has a list of agent identification (and their respective network address) and a list of skills available in the society (and the respective set of agents that is able to perform those skills).

As described earlier, a SACI agent should follow a predefined behavior. To *enter* into or *exit* a society, the agent should ask permission to the society facilitator. The SACI agent (including the facilitator) has an entity called *Mbox* (standing for Message Box) to queue the received messages for processing. An agent wanting entrance permission should send a message to the society facilitator that will capture the agent’s name and network address, and generate a unique identification for that agent in the society. This identification (and localization) is stored in a list and will provide a service of *white-pages* (just like in a phone catalog). To an agent leave the society, it will have to follow the same process, but the facilitator will, in its turn, remove the agent’s references from white-pages and yellow pages lists.

Once inside the society, SACI agents may exchange messages with each other and/or announce skills to the facilitator. To *exchange messages*, an agent (A) should ask the facilitator for localization of another agent (B). Once received this localization (network address from white-page list), the agent pair (A and B) can communicate directly exchanging KQML messages (see Figure 3).

The *announce* of agent skills obey the same process of entering into the society, but here the already set agent identification will be stored together with its specific skills in a list called *yellow-pages* that will provider the same service of a yellow-pages of a phone catalog where the subscribers are grouped according to their services. When an agent needs someone to do a specific job, it asks the facilitator for somebody who is able to perform it. After query the yellow-page list, the facilitator sends a list of agents back to the asker agent. By now on, the agent will ask the services directly to the provider agent.

Compared with some analogous tools, SACI presents a good performance [10]. This was checked counting down the messages sent between two agents (one providing a service and the other asking something to him) per a time unit. The tests were realized in 3 distinct situations:

- **S1**: Agents executing in the same JVM as threads
(Pentium 333MHz, 64MB RAM, Linux 2.2)

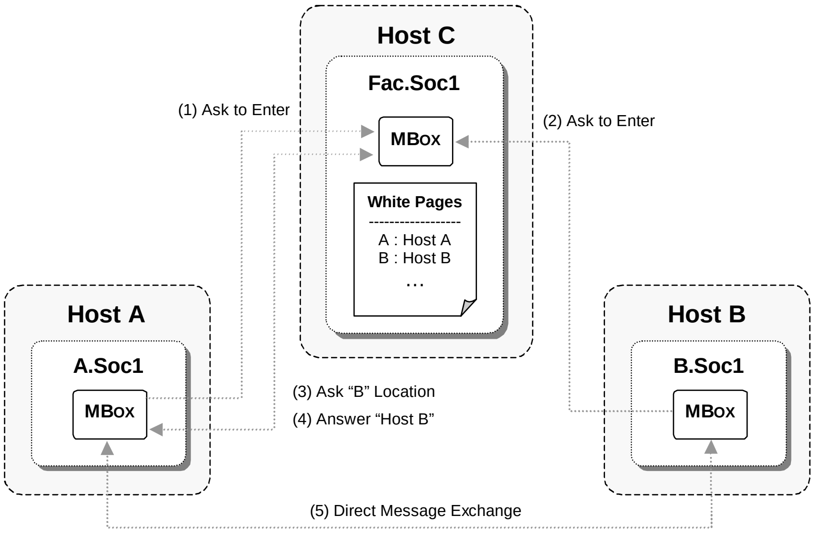


Fig. 3. White Page Service

- **S2**: Agents executing in distinct JVM but in the same machine
(Pentium 333MHz, 64MB RAM, Linux 2.2)
- **S3**: Agents executing distinct machines
(Server: Pentium 333MHz, 64MB RAM, Linux 2.2 /
Client: Pentium 233MHz, 128MB RAM, Linux 2.2.5)

The table below shows the *average* number of messages per second exchanged between the server and the client agents in the three situations described above:

Table 1. Performance Test Result

	TEST		
TOOL	**S1**	**S2**	**S3**
Saci 0.5	2412,54	197,24	137,49
Jackal 3.1	6,03	6,64	4,73
JKQML 5.1a	1,43	2,55	3,56
FIPA OS 1.1	17,95	25,13	18,86

4.2 SACI Extended Architecture

The limitations of the current handheld devices demand the presence of a server to manage the complex tasks of controlling the society. The SACI Extended Architecture (see Figure 4) is based on a client-server model where the server, implemented in J2SE and called SACI Server, manages the main functions of the system like society control. The clients, written in any language, use the services provided by this server.

The first step towards the extension of SACI to support handheld agents has been the implementation of a module responsible for listening HTTP requests, since it is the only transport mechanism available for handheld agents. This was implemented using a Java Servlet [38], which runs in an HTTP server. This Servlet gets all the requests, transforms them into SACI specific commands and apply them to the SACI Server. The responses of the commands are also send to the agents by this Servlet. Before this, SACI supported only RMI transport protocol not supported by J2ME. This new mechanism opens the SACI doors not only for the KSACI Agents, but for all agents implemented in any language that supports the HTTP protocol.

In this extended architecture, there are four possible kinds of clients: SACI clients running on a desktop, KSACI clients running on a handheld device, their respective proxy clients and the external clients from another tool. The Desktop SACI Client (written in J2SE), shown in Figure 4, as the name suggests, is the part of the SACI framework responsible for the behavior specification of the clients running on a desktop. The Handheld KSACI Client is implemented in J2ME and has a respective proxy in the desktop side to manage the messages between device and desktop, since the protocol used in SACI environment is different from the one supported by J2ME. This proxy client is also used for the "e*xternal*" clients (the ones written in another tool or language).

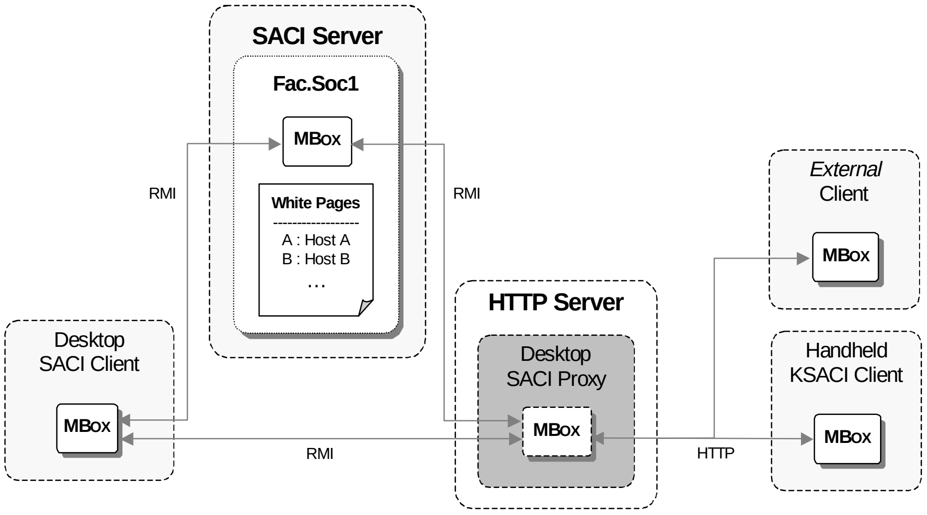


Fig. 4. KSACI architecture

The parts located in the desktop communicate with each other using RMI. When a Handheld KSACI Client or an External Client wants to communicate with one of these desktop parts, it has to send its messages through the Desktop SACI Proxy located inside a HTTP Server on desktop. This proxy is a special SACI Agent responsible for listen a TCP/IP port and redirecting the messages to the respective client or server on desktop via RMI. There will be a Proxy Client like this for each Handheld KSACI Client (and External Client), since a proxy will not be able to decide which handheld client to send the messages. The Handheld KSACI Client will have to be continuously checking the proxy Message Box (MBox) for new messages from other parts (facilitators and agents), since the only protocol available for J2ME is HTTP.

4.3 Transport Protocols

In the SACI Extended Architecture there are two kinds of transport protocols: RMI (Remote Method Invocation) and HTTP (Hipertext Transfer Protocol). The former is used in communications between the three parts written in J2SE (SACI Server, Desktop SACI Client and Desktop SACI Proxy). This protocol specifies that client Java applications can invoke methods on an object running in a remote server. The client uses a proxy object to represent the real, remote object. Because the proxy and the remote object implement the same interface, the application calls methods on the proxy as if it were directly calling methods on the remote object. RMI requires a Java Virtual Machine on both ends, and object serialization is used to pass parameters. This implies that a Desktop SACI Client can treat remote objects (located in another Desktop Client or in the SACI Server) as if it were present on the local VM.

The latter protocol (HTTP) uses a reliable TCP/IP connection. Handheld KSACI Clients are not then able to manage remote objects (see section 3) and it is necessary an alternative way to transport objects in the content of a message, since objects are very important in certain applications.

4.4 Agent Communication Language

For convenience, we choose KQML as our outer language for KSACI architecture since it was the ACL used by the current version of SACI and by several MAS nowadays. The future versions of SACI and KSACI will also provide support to FIPA ACL.

KQML is a language and a protocol specification to support high-level communication among agents [13;14]. And as an Agent Communication Language (ACL), KQML enables agents to exchange information and knowledge. As known, KQML does not specify its messages' content. As SACI uses Java as its language, the content language of the KQML messages can be either java Strings or serialized java objects. The java string can represent any declarative language, such as KIF and Prolog. In some application, it is natural to use a serialized object as the content language, since it is not necessary to have a parser on both sides of the communication. This last option is unfortunately not available for the KSACI Handheld Clients since there is no support to Java reflection and user class loaders, which forbids the application to load new classes other than those from the core or application *classpath*.

KSACI fully support string-based contents. As there is no Prolog or KIF-like interpreter already implemented in J2ME, the messages' contents tend to be unstructured or follow *ad hoc* syntaxes. To cope with this problem, we add to KSACI the capability of interpret XML contents. We claim that, in the context of handheld agents communication, XML [27;28] can play a major role as a content language because of three main reasons:

- Its data is self-describing in an internationally standardized and non proprietary format;
- It is catalyzing the convergence of some terms, which may be the initial steps for a world-wide ontology in some domains;
- It is possible to implement a simple and efficient XML parser in J2ME.

In order to provide XML contents exchange in KSACI, we have used kXML [33], a XML parser for J2ME, on the KSACI side. On the SACI server side, we have used a J2SE XML parser named Castor [32].

5 Related Works

This book presents two other tools to provide the communication components to handheld agents.

The first tool, Lightweight Extensible Agent Platform (LEAP) [15], is an adaptation of JADE kernel [36] to simplify the migration of legacy agents to a mobile network. LEAP is FIPA-compliant. It does not implement any module or component in J2ME, but provides compatibility of ‘‘PC agents’’ (written for JADE platform) and agents implemented in J2ME.

The second tool, MicroFIPA-OS [16], is an adaptation of FIPA-OS [37], also FIPA-compliant, for J2ME. The results of the performance evaluation of this adaptation on real handheld devices have not been satisfactory. However, the performance problems will be probably solved in a near future taking into account the evolution of the devices processing power and memory.

6 Conclusions

KSACI is a successful pioneer effort to enable agent communication capabilities in handheld devices, using J2ME. The KSACI Architecture provides an agent communication infrastructure for those who want to develop wireless applications for handheld devices. An agent located in one of these devices will be able to connect with a server and enter into an agent society, and communicate with other agents localized both in a PC or another handheld device. KSACI is already fully implemented and its preliminary test results, on cell phone emulators, are encouraging. Its main limitations are exclusively due to the current specifications of J2ME and related platforms and protocols.

We plan to include a support to FIPA ACL in KSACI architecture. We also intend to perform a systematic and quantitative evaluation of KSACI on real J2ME-based cell phones, and not only on emulators. As a long term goal, we plan to build an ensemble of APIs in J2ME covering the basic components of agent-oriented programming. In this sense, we about to finish the porting of JEOPS [5], an object-oriented production system, to J2ME, providing reasoning mechanism to handheld agents.

References

[1] Andrew C. Huang, Benjamin C. Ling, Shankar Ponnekanti, Armando Fox. “Pervasive Computing: What Is It Good For?. In proceedings of the Workshop on Mobile Data Management (MobiDE) in conjunction with ACM MobiCom '99, Seattle, WA, September 1999 (forthcoming)

[2] Java 2 Plataform, Micro Edition, http://java.sun.com/j2me

[3] Eric Giguère, “*Java 2 Micro Edition: The Ultimate Guide to Programming Handheld and Embedded Devices*”, Chapter 3. ISBN 0-471-39065-8.

[4] Java 2 Plataform, Standard Edition, http://java.sun.com/j2se/1.3

[5] JEOPS - The Java Embedded Object Production System, http://sourceforge.net/projects/jeops/

[6] Filho, C. S. F., Ramalho, G. L., JEOPS – The Java Embedded Object Production System, in Proc. Lecture Note in Artificial Intelligence no. 1952.

[7] JESS, http://herzberg.ca.sandia.gov/jess/

[8] InterProlog, http://www.declarativa.com/InterProlog/default.htm

[9] JavaLog, http://www.exa.unicen.edu.ar/~azunino/javalog.html

[10] Jomi Fred Hübner and Jaime Simão Sichman, “*SACI: Uma Ferramenta para Implementação e Monitoração da Comunicação entre Agentes*”, IBERAMIA, 2000

[11] Jomi Fred Hübner and Jaime Simão Sichman, “*SACI Programming Guide*”

[12] SACI Manual, http://www.lti.pcs.usp.br/saci/doc

[13] Yannis Labrou and Tim Finin, “*A Proposal for a new KQML Specification KQML*”,. UMBC, Baltimore, 1997.

[14] Yannis Labrou and Tim Finin, “*Agent Communication Language: the current landscape*”, IEEE Intelligent systems, March/April, 1999.

[15] Bergenti, F., Poggi, A., A FIPA Platform for Handheld and Mobile Devices, In Proceedings of ATAL’2001.

[16] Laukkanen, M., Tarkoma, S., Leinonen, J., FIPA-OS Agent for Small-footprint Devices, In Proceedings of ATAL’2001

[17] H. Haugeneder and D. Steiner. Co-operating agents: Concepts and applications. In N. R. Jennings and wooldridge, editors, Agent Technology Foundation, Application, and Markets, pages175-202. Springer-Verlag, 1998.

[18] Weiss, G. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, The MIT Press, Cambridge, Massachussets, London, England, 1999.

[19] RMI (Remote Method Invocation), http://java.sun.com/products/jdk/rmi/

[20] CORBA (Common Object Request Broker Architecture), http://www.corba.org/

[21] DCOM (Distributed Component Object Model), http://www.microsoft.com/com/tech/DCOM.asp

[22] Connected Limited Device Configuration (CLDC), http://java.sun.com/products/cldc/

[23] “*Applications for mobile Information Devices: White Paper*”, Sun Microsystems, Inc., 2000

[24] Wireless Technologies, http://developer.java.sun.com/developer/technicalArticles/wireless/#midp

[25] Mobile Information Device Profile (MIDP), *“http://java.sun.com/products/midp”*

[26] “*Java ™ 2 Platform Micro Edition (J2ME ™) Technology for Creating Mobile Devices: White Paper*”, Sun Microsystems, Inc., 2000

[27] The W3C XML Extensible Markup Language Working Group Homepage, http://www.w2c.org/XML

[28] Benjamin N, Grosof, Yannis Labrou and Hoi Y. Chan. “*A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML*”. In Proc. 1st ACM Conference on Eletronic Commerce (EC-99), Denver, Colorado, USA, 1997, http://www.ibm.com/iac/ec99/

[29] Alvares, L. O., Sichman, J. S. "*Introdução aos sistemas multiagentes*". In: MEDEIROS, C. M. B. (Ed.) Jornada de Atualização em Informática. Brasília: SBC, agosto 1997. v. 16, Cap. 1, p. 1ss.
[30] Big in Japan, http://www.javasoft.com/features/2001/03/docomo.html?frontpage-banner. Visited on march-29-2001.
[31] All about I-mode, http://www.nttdocomo.com/pr/recommend/d503i.html. Visited on april-01-2001
[32] Castor, http://castor.exolab.org/
[33] kXML, http://www.kxml.org/
[34] TinyXML, http://www.gibaradunn.srac.org/tiny/index.shtml
[35] NanoXML, http://nanoxml.sourceforge.net/
[36] Bellifemine, F., Rimassa, G., Poggi, A., JADE - A FIPA compliant Agent Framework. Proc. of the 4th International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents, London, 1999.
[37] FIPA-OS, http://fipa-os.sourceforge.net/
[38] Servlets, http://java.sun.com/products/servlet/

LEAP: A FIPA Platform for Handheld and Mobile Devices

Federico Bergenti and Agostino Poggi

Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Parma,
Parco Area delle Scienze 181A, 43100 Parma, Italy
{Bergenti,Poggi}@CE.UniPR.IT

Abstract. The ever-increasing importance of the market of portable devices is promoting the migration of technologies originally developed for the fixed network to the mobile network. This paper describes the general aim and the current results of a European-scale project intended to provide the enabling technology for deploying multi-agent systems across fixed and mobile networks. The LEAP project achieves its goal realising a FIPA platform that can be deployed seamlessly on any Java-enabled device with sufficient resources and with a wired or wireless connection. Such a platform is implemented as a new kernel for JADE to ease the migration of legacy agents to the mobile network and it exploits a modular design to scale its functionality with the capabilities of the device.

1 Introduction

The enormous increase of the market of mobile devices in recent years is stimulating the research community to port almost any technology developed for desktop computers to the realm of portable devices connected to a mobile network. This work anticipates the future needs of VHE – Virtual Home Environment – and it will make the step toward ubiquitous computing easier. Researchers in the field of agent technology follow this trend migrating agents and multi-agent systems to mobile devices. Nevertheless, we believe that the technology succeeding in enabling this migration will have to:

1. Conform to international standards, to ease the deployment of applications capable of exploiting third-party services running on both the fixed and the mobile network;
2. Allow agents adapting their characteristics to the capabilities of the device, to support users in accessing the same application, with eventually different characteristics, from a desktop computer, a palmtop or a cellular phone.

This paper describes the effort spent and the current results of the *LEAP – Lightweight and Extensible Agent Platform* – project in providing the basic technology for running FIPA [3] agents seamlessly on any Java-enabled device with sufficient resources and with a connection to a fixed or to a mobile network. The project LEAP is a European-scale effort funded for 2.5 years by the European Commission and it involves research centres spread across Europe in: France (Motorola), Germany (Siemens and ADAC), Ireland (Broadcom), England (BT) and Italy (TILAB and Università degli Studi di Parma). The result of the LEAP project is LEAP, a FIPA platform that can be used to deploy multi-agent systems spread across a heterogeneous network of mobile and fixed devices, ranging from cellular phones to enterprise servers. We released version 1 of LEAP to registered users for testing purposes, and version 2 will be released in open source by the end of this year. The functional characteristics of LEAP version 1 are:

1. It runs seamlessly on desktop PCs and on handy devices with limited resources, such as Palm Pilots and emulated Java-enabled phones;
2. It exploits wireless networks, such as TCP/IP over GSM and IEEE 802.11 wireless LAN, for guaranteeing connectivity to handy devices;
3. It is operating-system agnostic, i.e., it exploits Java to achieve full portability also on operating systems for small devices, such as Microsoft Windows CE and PalmOS.

In order to maximise the acceptance of this new platform, we decided to start its implementation from an existing platform available in open source with the following characteristics:

1. It must have a sufficiently modular design to support a downsizing process;
2. It must have a consolidated community of users interested in implementing ubiquitous multi-agent systems;
3. It must have an accepted and stable API;
4. It must not rely on third-party pieces of software that might not be downsized.

We believe that JADE [1] fulfils such requirements [11, 14], and LEAP is designed and implemented as a new kernel for JADE. The result of this development approach is that any legacy agent developed with JADE can now run in a wireless environment provided that the device offers sufficient resources and processing power.

The rest of this paper is organised as follows: in section 2 we describe in greater detail the objectives of the LEAP project. Section 3 shows the architectural design of the platform and gives some hints on its implementation. Finally, section 4 outlines some conclusions and future work.

2 Objectives of the LEAP Project

Agents need resources to act and to communicate and the agent platform is the run-time support providing such resources. Agents can run only in the scope of an agent platform that provides the basic services to support interoperability, i.e., a means for sending and receiving messages and a means for locating agents. The platform is not responsible for providing any means for implementing service-level interoperability or autonomy [2] and it must be completed with other components to support the concepts of agent-oriented software engineering, such as roles, goals and plans [8, 10].

Agents communicate explicitly sending messages and such messages may reach either agents within the same platform or agents on different platforms. This difference must be transparent to the developer and the platform is responsible for this. In addition, an agent platform can be physically distributed across a set of network nodes and we call such nodes containers because they are meant to host agents belonging to the platform.

The distribution and cooperation of agents residing on different platforms implies the conformance to a standard. At the moment, only FIPA is producing specifications for agent platforms. These specifications provide an abstract architecture describing the services of an agent platform [4] as well as a specification for the aspects related to the lifecycle of agents and platforms [5]. FIPA has just released FIPA 2000 specifications [3] and the aspects of FIPA 2000 relevant to the LEAP project are:

1. Agent management and agent communication: support for a variety of encoding and naming schemes, including support of agents in mobile contexts;
2. Nomadic computing: agent management and communication frameworks geared towards support of mobile communication networks.

At the moment, a number of FIPA platforms are available [1, 7, 9, 12, 13], but none of them is capable of running on mobile devices with restricted resources. The LEAP project is meant to fill this lack and the following is the list of its concrete objectives:

1. Devise and implement a FIPA compliant platform capable of deployment on both fixed devices, such as workstations and desktop computers, and mobile devices, such as PDAs and cellular phones;
2. Realise an operating-system agnostic platform, capable of deployment on several operating systems including: Microsoft Windows CE, Palm OS, EPOC;
3. Ensure that the platform can operate over both a fixed or a wireless network, so that application using it can run seamlessly in both environments;
4. Manage the ability to configure the platform with respect to the capabilities of a target device.

In addition to the enabling technology for taking agents to the mobile network, the LEAP project will develop three generic agent services designed to show the possibilities of agent technology in assisting people irrespective of physical location, time of day, day of week, etc. In particular, we address the needs for open infrastructures and services supporting dynamic enterprises and mobile teams. The characteristics of these services are that they are able to:

1. Anticipate a user's knowledge needs by accessing and customising knowledge on the basis of the user's profile and location;
2. Provide access to collective knowledge assets in the team by networking individuals with each other, based on their current needs;
3. Empower individuals to collectively coordinate activities, e.g., by trading jobs, automatically negotiating for work, and expressing personal preferences;
4. Anticipate a user's travel needs, providing guidance and time estimation so as to synchronise the movements of virtual teams working over geographic areas.

The products of the LEAP project will be validated in two field trials targeted at mobile engineers working in the telecommunications and roadside assistance industries. This will provide an insight into the practicalities and issues related to the industrial deployment and management of agent services on mobile devices. Additionally, the field trials will provide a means for investigating the social and usability aspects of agent services in an operational environment. These trials should clarify the importance of the results obtained by the project and should also show how agent technology could play a central role in the development of customer-oriented services integrating the possibilities offered by the fixed and the mobile networks.

Figure 1 shows a coarse-grained diagram of the components involved in an application built on top of LEAP. Java hides the differences among devices concerning the operating system and the network. LEAP provides the services for agents to execute and to communicate. The generic agent services implement a framework for the integration of the application logic intended to realise the functionality of the application. Such services will be implemented during this year and they will be used to implement the applications for the field trials. Finally, the GUIs allow the user interacting with the agents and with the platform.

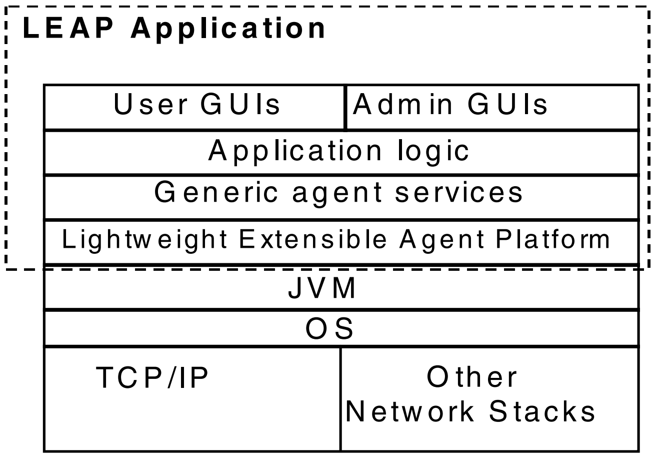


Fig. 1. Coarse-grained components of an application built on top of LEAP

3 Architectural View of the LEAP Platform

The main goal of the LEAP project is to develop a FIPA-compliant agent platform sufficiently lightweight to execute on small devices but also sufficiently powerful, flexible and open to be a first-class choice for enterprise servers. To satisfy this goal, LEAP can be deployed according to a set of profiles identifying the functionality available on each particular device. The basic profile supports only the functionality required by FIPA compliancy and it suits the smallest device that we address, i.e., a cellular phone. The full-featured profile provides the functionality available on an agent platform designed to run on PCs and it copes with any device with sufficient memory and processing power. The choice of implementing LEAP as a lightweight and extensible kernel for JADE allows using the services that JADE offers across all profiles. In the current implementation of LEAP, the basic profile provides the subset of JADE APIs supporting FIPA compliancy and it does not integrate any run-time tool, such as the RMA or the Sniffer [1]. On the contrary, the full-featured profile integrates all tools that JADE provides and, from the developer point of view, it offers the same functionality and the same APIs of JADE. All profiles are instantiations of the FIPA abstract architecture and agents running on platforms configured with different profiles can interoperate. The architecture of the platform is modular and it is organised in terms of:

1. A mandatory kernel module, that manages the platform and the lifecycle of agents;
2. A mandatory communication module, that handles the heterogeneity of communication protocols;
3. An optional administration GUI module, that allows users managing the lifecycle of agents, visualising agents and message exchanges, and sending messages to agents.

The minimal set of the two mandatory modules is sufficient to run a fully operational platform on the most limited target device.

The introduction of the concept of profile in JADE required a substantial redesign and re-implementation of its internals but we succeeded in realising it without changing the APIs. As a consequence, the community of JADE users can run existing applications on small and wireless devices without any modification of their code, provided that the specific device offers sufficient resources.

3.1 Deployment Issues

The design of LEAP shares the basic principles of the design of JADE. Nevertheless, its implementation is basically different from JADE because the latter was not thought taking into account the limitations of small devices. The devices on which LEAP is supposed to run vary widely in terms of memory, computation, display, pointing devices and connectivity capabilities, as shown in table 1.

Table 1. Target devices for the LEAP platform

Device category	Memory	User interface	Connectivity
Cellular phone	< 16 Mbytes	About 10 lines on the screen	1. TCP/IP over GSM 2. SMS
PDA	16-32 Mbytes	Small screen (some tens of lines and columns)	1. TCP/IP over GSM 2. SMS 3. IEEE 802.11
Workstation	64-256 Mbytes	Monitor, keyboard and mouse	1. TCP/IP

It is worth noting that the resources available to Java applications can vary significantly from nominal values shown in table 1, mainly because of bugs in the implementations of the virtual machines. For example, the implementation of the KVM for PalmOS restricts to 200kbytes the heap memory available to Java applications even if the device has 8Mbytes.

LEAP adapts to target devices while maintaining a minimal footprint; hence the architecture of the platform needs to be modular. In particular, the modules that compose LEAP architecture are categorised into:

1. Mandatory and device independent, like the agent management module;
2. Mandatory and dependent from the capabilities of the devices, such as the communication module;
3. Optional and device independent, such as the security plug-in for the communication module;
4. Optional and dependent from the capabilities of the devices, such as all GUIs.

A module may not implement completely a given high-level functionality. For instance, the ACC is split over agent management and communication modules.

In aiming to make LEAP as universal as possible, we designed it to be operating system agnostic and to work on a variety of communication protocols. We decided to base our development on the Java 2 platform in order to access a common layer of platform-independent functionality available on mobile phones, PDA and PCs running all sorts of operating systems. Even if Java provides a solid foundation to build an agent platform running seamlessly on different target devices, some device-specific customisations are necessary, as small devices do not provide a full-featured Java 2 platform. Considering the diversity of existing devices, Sun decided [15] to group Java platforms into three editions, each having its proper Java virtual machine:

1. Java 2 Enterprise Edition (J2EE), intended and optimised for enterprise servers and mission-critical applications;
2. Java 2 Standard Edition (J2SE), intended for PCs and workstations and optimised for ordinary applications;
3. Java 2 Micro Edition (J2ME), intended for devices with restricted memory and processing power, such as Java-enabled phones and PDAs.

Moreover, J2ME introduces the notion of configuration. A configuration of J2ME specifies a subset of its Java virtual machine features and an associated subset of Java programming language features. There are currently two configurations within J2ME:

1. Connected Device Configuration (CDC), for devices with memory and processing power comparable to a small PC;
2. Connected, Limited Device Configuration (CLDC), for connected devices with strict restrictions on resources.

Most of the functionality in CLDC and CDC has been inherited from J2SE. In addition, CLDC and CDC introduce a number a features, not drawn from the J2SE, designed specifically to fit the needs of small devices. Such features refer mainly to display and communication.

LEAP is not only meant for small devices, it is also intended to support enterprise servers and we cannot impose constraints on its functionality when running in environments with no restrictions on resources. Therefore, we decided to go for the worst case, i.e., the CLDC, implementing an extensible architecture over a layer of adaptation capable of matching the classes available on this platform with the ones available on other Java 2 platforms. The following subsection shows an example of the layer of adaptation that we implemented for the specific case of handling multiple transport protocols.

3.2 Communication Issues

LEAP is naturally distributed as it is composed of a number of agent containers. We impose no restrictions on the way containers can be deployed across the nodes of the network but the best way of deploying the platform is having one container running in one Java virtual machine for every node. LEAP allows deploying containers belonging to the same platform according to different profiles. It is worth noting that splitting the platform into several containers distributed on different devices is not mandatory. According to the context and the requirements of a given application, it is possible to concentrate the whole platform into a single container or to distribute it across the nodes of the network. As an example, if the application consists of a personal assistant supporting the user in managing the information on her PDA, probably the best deployment choice is to have a single-container platform running on the PDA. On the contrary, given an application where each user in a team is assisted by an agent running on her mobile phone and all such agents interoperate between each other and with a coordination centre, the choice of a distributed platform composed of one container running on an host connected to the Internet and several peripheral containers on the users' phones can be the best solution.

The choice of spreading the platform across the network poses some problems for agents running on other platforms in communicating with agents running on a LEAP platform. Following the design of JADE, we solved this problem introducing a privileged container, called main container, to allow agents on other platforms considering LEAP as a whole. The main container is unique and acts as a front-end for the platform. It maintains platform-wide information and provides platform-wide services. This container must always be reachable by all active containers in the platform. A deployment of LEAP is composed of a single main container and a set of peripheral containers, allowing a high modularity by running lightweight peripheral containers on small devices. The main container provides the FIPA mandatory services, i.e., the AMS and the DF, and the amount of resources that they need in the current implementation suggests that the main container should run on a fixed network node with no particular restrictions on processing power and memory.

FIPA specifies a set of communication protocols that can be used to send messages to agents and it specifies how a gateway for matching such protocols should be [6]. Nevertheless, the first FIPA specifications did not allow many communication protocols, but they prescripted that all FIPA agents were addressed through a CORBA interface. Therefore, the first implementations of FIPA platforms relied on IIOP for communication between agents and, where applicable, this choice is still adopted. The main container of LEAP is supposed to run on a PC with no restrictions on the available resources and we can reasonably suppose that we could find an implementation of IIOP there. This assumption is not valid for containers running on mobile devices and it may not be the best solution also for PCs. For example, the container may run behind a firewall and direct IIOP communication could be impossible or it may require sending Java objects to the main container and Java RMI could be a better solution. Therefore, we need to support the communication between the main container and the rest of the platform by means of a flexible mechanism allowing the integration of different communication protocols. We call such protocols ITP – Internal Transport Protocol – and the current implementation of LEAP provides the following ITPs:

1. IIOP;
2. Java RMI;
3. Proprietary protocol over TCP/IP for wired and wireless connections.

Only the last protocol is a feasible choice for small devices connected to the wireless network, even if implementations of CORBA for wireless networks are available. We used such a protocol to support the communication between a palmtop, the Compaq iPAQ H3600, and a fixed network node exploiting both TCP/IP over GSM and IEEE 802.11 wireless LAN.

FIPA introduces the AMS as the authority governing the platform and it requires agents consulting it to be granted in lifecycle transitions. FIPA does not specify how agents should interact with the AMS for management operations because these activities are private to the platform and each platform is allowed to use its optimised techniques. LEAP exploits the availability of many ITPs to support internal communication with the AMS. In particular, we use distributed-object technology where available, while we rely on our proprietary protocol where more sophisticated techniques cannot be applied.

As far as the communication mechanisms are concerned, containers must be able to:

1. Send ACL messages through at least one ITP;
2. Accept ACL messages through at least one ITP;
3. Buffer incoming messages directed to one of its agents;
4. Forward ACL messages directed to an agent running in another container to its main container;
5. Send management messages to the main container;
6. Accept management messages from the main container.

For profiles of the platform where no particular constraints on resources are present, containers also exhibit the following functionality:

1. Forward ACL messages directed to an agent running in another container through direct connection with the peer container. This requires the possibility of creating a direct channel between the containers and therefore it requires both containers having at least one common ITP;
2. Cache of addresses, a container might decide to keep an association between the address of an agent and the address of the containers where it runs. This could be used as a cache promoting the efficiency of the direct connection between containers.

Figure 2 shows a possible deployment of LEAP and emphasises the used communication mechanisms. It shows three containers: the main container runs on an enterprise server connected to the Internet, container 1 runs on a desktop computer and container 2 runs on a Java-enabled mobile phone. Containers are connected between each other through a feasible ITP, i.e., an ITP that, for each pair of container, is available on both containers. If, for some pairs of containers, no feasible ITP is available, then the main container acts as a router for internal messages. Only the containers supporting IIOP, i.e., the main container and container 1, are capable of connecting directly with other FIPA platforms. For the case of container 2, any message to and from other platforms is routed through the main container.

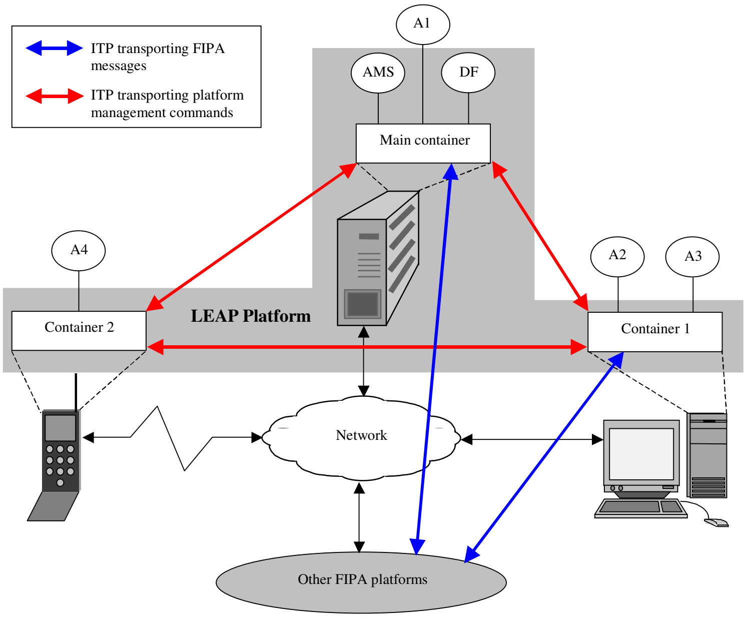


Fig. 2. Transport mechanisms in LEAP

LEAP needs to transport messages using different protocols. For this reason it is useful to introduce an abstraction layer hiding the details of the underlying transport mechanisms. The singleton object of class `AgentToolkit` acts as the unique entry point for the messaging mechanism, hiding transport details. Agents exchange ACL messages using only such an object, but the platform needs to manage many communication channels supporting different MTPs – Message Transport Protocols. The class diagram shown in figure 3 describes the solution we implemented emphasising the FIPA standardised MTP, i.e., fipa-iiop-std, and LEAP specific MTPs. The concrete subclasses of the class `MTP` provide the concrete implementation of a protocol exploiting a mono-directional instantiation of the Half Object Plus Protocol Pattern. This pattern allows providing a single interface for accessing different protocols in terms of proxies of remote objects. The concrete protocol used to marshal, send and receive the information is encapsulated in the concrete subclasses of both the proxies and the stubs.

4 Conclusions and Future Work

The impressive progress of the market of portable devices allows delegating increasingly complex tasks to small, wireless and ubiquitous devices.

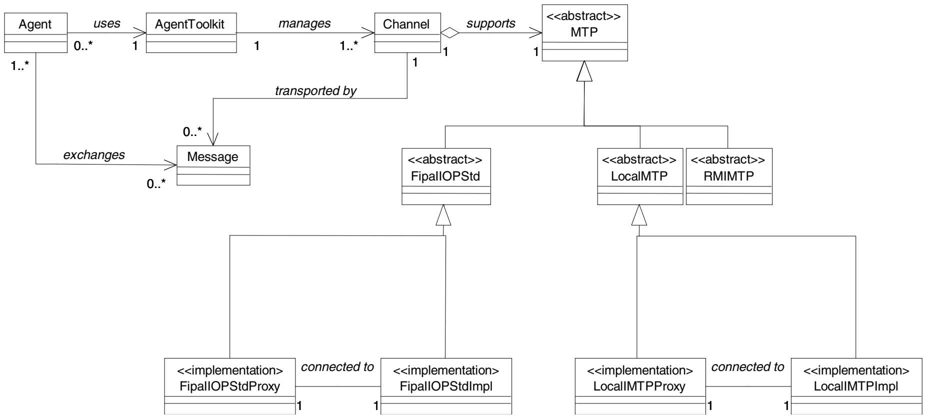


Fig. 3. LEAP protocol-agnostic messaging

This trend promotes the porting of technologies originally developed for PCs and workstations to mobile devices.

This paper follows this trend describing the current results of the LEAP project. This project is intended to migrate agent technology to mobile devices through the development of a FIPA agent platform running seamlessly on any Java-enabled device, from cellular phones to enterprise servers.

The project has released the first version of the platform and the second, and final, version will be released in open source by the end of the year. The work toward version two of the platform is intended to:

1. Optimise the current implementation with particular regards to the transport mechanism for wireless networks;
2. Improve the support for wireless connections exploiting the availability of new protocols such as GPRS and Bluetooth;
3. Continue the downsizing of the platform to meet the requirements of real cellular phones;
4. Handle more efficiently the disconnected state.

We believe that the LEAP project opens new fields of applications to agent technology, where location awareness and team work can render new services to users by providing accurate information linked with their physical environment as well as with the dynamic formation of teams of people or with the dynamic formation of services answering to personal and specific needs.

Acknowledgement

The authors would like to thank the other partners of the LEAP project: ADAC, Broadcom, BT, Motorola, Siemens and TILAB. The European Commission funds this work through the grant IST-1999-10211.

References

1 F. Bellifemine, A. Poggi, and G. Rimassa. “Developing Multi-agent Systems with a FIPA-compliant Agent Framework”, in Software – Practice and Experience, Vol. 31, pp. 103-128, 2001.
2 F. Bergenti and A. Poggi, “A Development Toolkit to Realize Autonomous and Interoperable Agents”, in Proceedings of Autonomous Agents 2001, 2001.
3 FIPA “FIPA 2000 Specifications”, available at `http://www.fipa.org`.
4 FIPA “FIPA Abstract Architecture Specification”, available at `http://www.fipa.org`.
5 FIPA “FIPA Agent Management Specification”, available at `http://www.fipa.org`.
6 FIPA “FIPA Wireless Message Transport Protocol Specification”, available at `http://www.fipa.org`.
7 J. Heecheol, C. Petrie and M. R. Cutkosky. “JATLite: A Java Agent Infrastructure with Message Routing”, IEEE Internet Computing, Mar./Apr., 2000.
8 C. A. Iglesias, M. Garijo and J. C. A. González, “Survey of Agent-Oriented Methodologies”, in Proceedings of ATAL’98, 1998.
9 F. G. McCabe, “April – An Agent Programming Language for the Internet”, available at `http://www.nar.fujitsulabs.com`.
10 MESSAGE Consortium, “Deliverable 1: Initial Methodology”, deliverable of the EURESCOM Project P907-GI, 2000.
11 M. Nawostawski, G. Bush, M. Purvis and S. Cranefield. “Platforms for Agent-Oriented Software Engineering”, in Proceedings of APSEC 2000, pp. 480–488, 2000.
12 H. S. Nwana, D. T. Ndumu and L. C. Lee, “ZEUS: An advanced Toolkit for Engineering Distributed Multi-Agent Systems”, in Proceedings of PAAM’98, London, 1998.
13 S. Poslad, P. Buckle and R. Hadingham, “The FIPA-OS Agent Platform: Open Source for Open Standards”, available at `http://fipa-os.sourceforge.net`.
14 P. M. Ricordel and Y. Demazeau. “From Analysis to Deployment: A Multi-Agent Platform Survey”, in Omicini A., Tolksdorf R., and F. Zambonelli (Eds.) Engineering Societies in the Agents World, pp. 93-105, Springer-Verlag, 2000.
15 Sun Microsystems, “Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices”, available at `http://www.java.sun.com`.

FIPA-OS Agent Platform for Small-Footprint Devices

Mikko Laukkanen[1], Sasu Tarkoma[2], and Jani Leinonen[2]

[1] Sonera Corporation
P.O.Box 970 (Teollisuuskatu 13), FIN-00051 SONERA
Helsinki, Finland
mikko.t.laukkanen@sonera.com
[2] University of Helsinki
P.O.Box 26 (Teollisuuskatu 23), FIN-00140 University of Helsinki
Helsinki, Finland
{sasu.tarkoma,jani.leinonen}@cs.helsinki.fi

Abstract. The trend is towards having smaller and smaller mobile devices, also called small-footprint devices, allowing nomadic users to access the same services as with the static computers from virtually anywhere and at any time. The idea of having software agents running on small-footprint mobile devices sounds an attractive way of delivering services for nomadic users. One of the biggest problems is that the current agent platforms are designed to run on computers with a lot of resources, e.g., CPU power and memory. This paper discusses the problem area of having a FIPA-OS agent platform running on a small-footprint devices. Our views are based on experiences of running a FIPA-OS agent platform on Java-enabled small-footprint devices. The experiments were conducted using Casio Cassiopeia E-115, Psion Series 5mx and Compaq iPAQ H3630. Our results clearly show that without any optimizations FIPA-OS is not suited to run on small-footprint devices.

1 Introduction

Nomadic computing is an emerging technology enabling access to the fixed network services, such as Web and e-mail, from virtually anywhere and at any time [22]. Software agent technology on the other hand is assumed to be a design methodology for implementing complex distributed systems [15]. A combination of these two technologies offers a possibility to create software supporting nomadic users and running on small-footprint devices, such as a personal digital assistant (PDA) or mobile phone. We call this kind of technology agent-based nomadic computing.

However, there are challenges in making the agent-based nomadic computing a reality. The end-devices have to be small, if they have to be carried by the user. Also, the communication usually happens over a low-bandwith wireless link. Thus, the use of wireless link should be optimized.

The former challenge of having small portable devices has become reality as the current generation of PDAs is almost pocket sized and the devices are powerful enough to execute middleware services, such as Java and CORBA. The latter challenge, on the other hand, is still seeking answers. However, the communication aspect of agent-based nomadic computing is out of the scope of this paper.

The motivation for having a middleware layer, such as Java or a software agent execution environment, on small and wireless devices is portability, interoperability and the ease of development. Java language is portable between a wide range of environments with the expense of reduced performance during execution. Also, the object-oriented approach of Java focuses on the reuse of components. Furthermore, agents, which are often implemented using Java, offer interoperability with a wide range of both agent-based and non-agent-based services and increased flexibility due to the behaviour-based programming paradigm [18].

Intelligent agents and applications usually correlate with high computing power and are, up until now, designed to run on high-performance desktop or laptop computers. It is possible to execute agents on current PDAs', however, the small-footprint agent execution environment needs to be very lightweight and scalable to a range of different CPU speeds and hardware configurations. This paper addresses the problems related to having agent platforms running on small-footprint devices.

FIPA (Foundation for Intelligent Physical Agents) [13] is a standardization organization promoting development and specification of agent technologies. The main goal for FIPA is to specify how different kinds of agent platforms can interoperate. FIPA specifications have reached such a maturity that FIPA-compliant agent platforms have been implemented by various developers. FIPA-OS [16,19] and JADE [2] are examples of open-source implementations of the mandatory elements of a FIPA-compliant agent platform.

This paper is organized as follows. Section 2 gives a motivation and describes the problem areas of agent-based computing. In Section 3 the test environment in terms of small-footprint devices and the FIPA-OS agent platform is described. In Section 4 we introduce a number of performance tests that were conducted to identify problems in running FIPA-OS on small-footprint devices. In Section 5 we discuss the results and give our proposals on how FIPA-OS could be improved and optimised for small-footprint devices. Finally, section 6 concludes this paper.

2 Problems in Running Java Agent Platform on Small-Footprint Devices

In order to combine Java and software agent technology for increased portability and interoperability several problem areas have to be considered: limitations of the small-footprint devices, limitations of the Java environment in small-footprint devices and the portability of FIPA-OS onto small-footprint devices. Each of the problems is discussed in upcoming subsections.

2.1 Limitations of Small-Footprint Devices

Small-footprint devices are originally meant to be personal assistants providing user with simple applications such as agenda, calendar and phone-book. However, the trend is towards having the small-footprint device as a “multimedia companion” allowing the user to access Internet, watch multimedia streams and listen to CD-quality audio.

At the time of writing this paper, high-end small-footprint devices usually have 32MB of RAM. In addition they usually have 16-32MB of ROM, on which the operating system and pre-installed application software is installed to.

The screen size and resolution are limitations, which will always be present with small-footprint devices and have to be taken into account when desiging GUIs, because the GUIs designed for standard monitor resolutions[1] cannot be adopted as such to the small-footprint devices [29]. Furthermore, small-footprint devices without keyboards usually come with a touchscreen as the input method. Applications on this kind of devices cannot rely only on textual input, because writing with this kind of devices is time-consuming compared to using a keyboard.

Connectivity with small-footprint devices is still a limiting factor, but has evolved greatly along with the introduction of new devices. Before small-footprint devices were meant only for synchronizing with the a desktop PC and possibly accessing Internet using a serial cable attached to a cellular phone. Nowadays the devices provide many ways for networking. In addition to synchronizing with desktop PC, IrDA [1] can be used for interacting with a cellular phone and having a point-to-point connection with another device. A serial port can be used in connecting many kinds of devices, such as cellular phones, digital cameras and GPS devices. There is also a large variety of different kinds of CF Card modems and both wired and wireless network interface cards available. Some devices also provide PC Card slots, which enable the use of a vast amount of PC Cards available.

One of the largest problems with small-footprint devices is the power consumption. For instance if WLAN card is used for accessing the wireless network, the power consumption is over double the amount of not having a WLAN-card inserted to the device [21].

2.2 Java on Small-Footprint Devices

There are currently several Java solutions for creating Java software for small-footprint devices. The Java 2 MicroEdition (J2ME) [26] is a scalable architecture for developing applications for consumer devices. J2ME consists of different configurations that define the Java Virtual Machine (JVM) and the minimal Java packages. Profiles define additional APIs for a particular configuration. Sun and its industry partners have released two configurations for different types of devices: the Connected Device Configuration (CDC) for higher end PDAs and

[1] By standard resolutions we mean resolutions, which are at least of size 1024x768.

Connected Limited Device Configuration (CLDC) for medium to low level PDAs and smart phones. CDC is based on the standard JVM and CLDC on the more limited one called KVM [25]. Code written for the KVM is less portable than the code written for Java standard edition virtual machine. Neither does KVM support hardware programming through Java Native Interface (JNI).

PersonalJava application environment is targeted at specific device platforms. With the PersonalJava runtime environment it is possible to run applets and applications written to the appropriate PersonalJava API specification [27]. PersonalJava will be integrated into CDC as Personal Profile. PersonalJava is portable between a wide range of devices, because many companies provide PersonalJava Virtual Machines and PersonalJava provides approximately JDK 1.1 functionality with restrictions depending on the implementation.

2.3 FIPA-OS in Small-Footprint Devices

The current FIPA-OS codebase is designed for standard workstations with fixed Internet connections and large amounts of persistent storage. The codebase is available for both JDK 1.1[2] and JDK 1.2. The system employs FIPA-ACL [9,10], FIPA-SL0 [12] and several other parsers generated using the JavaCC [17] tool and the Xerces XML parser from the Apache Software Foundation [14].

FIPA-OS agents use tasks to create concurrent behaviours that are used, for example, to handle messaging. Thread-based task management does not scale well to small-footprint devices, because it creates excess overhead and builds up latency in the system. Undeterministic garbage collection is one of the problems of Java in general [7], and this has to be taken into account when developing software for small-footprint devices. Garbage collection can be helped by reusing unused objects [3], using scalar types instead of objects, using lazy instantiation and avoiding string concatenation.

In addition, internal agent communication is done using Java RMI, which also poses extra overhead in constrained environments. In order to have the best possible performance, the agent communication should be based on method invocations and passing Java objects, if possible.

3 Test Environment

3.1 Target Devices

The objective of the tests is to find out, how much the performance of FIPA-OS running on different end-devices differ from each other. Therefore, the tests were conducted with identical codebase on devices differing from each other in terms of CPU power and available runtime memory and storage space.

As a reference we used a desktop computer, which roughly corresponds to the usual running environment for FIPA-OS. For the small-footprint devices

[2] At the time of writing this paper, the support for Java 1.1 is being added to FIPA-OS.

we choose to use Psion Series 5mx, Casio Cassiopeia E-115 and Compaq iPAQ H3630. Psion comes with the EPOC 5 [28] operating system and includes Symbian's JDK1.1.4-compliant Java runtime enviroment. Cassiopeia is a PocketPC device, which supports Sun's PersonalJava 1.0 compliant Java runtime environment, which corresponds to JDK1.1.x [27]. Compaq iPAQ runs Linux and Sun JDK1.1.8. In Table 1 the features of the devices are presented.

Table 1. The features of the devices used in the tests

Feature	Psion	Casio	iPAQ	Desktop
CPU	36MHz ARM	131MHz MIPS	206MHz ARM	500MHz PIII
RAM	16MB	32MB	32MB	256MB
Storage	10MB	16MB	32MB	9GB
OS	EPOC 5	PocketPC	Linux 2.4.3	Linux 2.2.16
JVM	EPOC JVM 1.1.4	Sun PJava 1.0	Sun JDK 1.1.8	Sun JDK 1.1.6

3.2 Modifications to FIPA-OS

In the tests we used FIPA-OS version 1.3.0, which was built only on JDK 2. Because the available JVM implementations for small-footprint devices only support Java 1.1, modifications were done to the code in order to make it run on the small-footprint devices used in the tests.

Most of the modifications were about replacing the Java 2 Collections with Java 1.1 structures. Although the Collections libraries are also available for Java 1.1, we decided not to use these, as the amount of storage space on small-footprint devices is limited. Thus, what we did was to change the Collection classes to either Hashtables or Vectors, depending on what kind of functionality was needed.

One difference between Java 2 and Java 1.1 is that the Java 1.1 lacks inbuilt CORBA ORB, which in turn is used by FIPA-OS for communicating with other platforms. In our tests we concentrated on intra-platform functionality, but for FIPA-compliance, we replaced the Sun IIOP message transport protocol (MTP) with HORB [24], which is a pure Java implementation of CORBA ORB.

In FIPA-OS the RMI is used for agent communication within one platform. As the RMI differs between Java 2 and Java 1.1 [23], we had to recompile the RMI MTP of FIPA-OS, but no major modifications were needed. When the FIPA-OS is started up, the RMI naming service is started up first and this is usually done independently before the agent loader starts to load the agents. This results in that the RMI naming service runs on its own JVM. In our tests, we used a small bootstrap class, which took care of starting everything up within a single JVM.

4 Performance of FIPA-OS on the Small-Footprint Devices

4.1 CaffeineMark Tests

CaffeineMark [4] is a benchmark program, which measures the performance of the execution of Java programs. In Figure 1 the CaffeineMark 3.0 benchmark results for the target devices are presented. These results can be used as a reference for our tests. From the results it can be seen that the devices can be divided into three categories based on the performance: Casio and Psion forms are quite equal, whereas iPAQ and Desktop form their own categories.

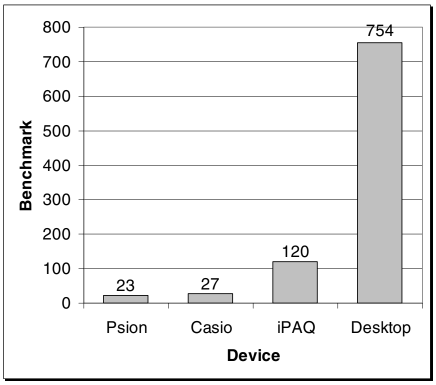


Fig. 1. CaffeineMark benchmark results for the target devices. Larger bar implies better result

4.2 Platform Startup Time

In the first test case we measure the FIPA-OS startup time. The whole process consists of starting three major parts: RMI naming service (NS), Agent Management System (AMS) and Directory Facilitator (DF). For more information about AMS and DF, see [8]. In Figure 2 the results for the platform startup are depicted. It must be noted that in Figure 2, the cumulative time is shown. Therefore, the time values for each of the parts do not indicate the times needed for the corresponding parts to be started up. Instead, the time values imply that the corresponding time is elapsed from the beginning of the platform startup.

Result show that although platform startup is done usually rearly, performance of Casio and Psion are not at acceptable level, whereas iPAQ's is quite acceptable; waiting ten seconds for an application to start is not unusual even with applications in desktop computers.

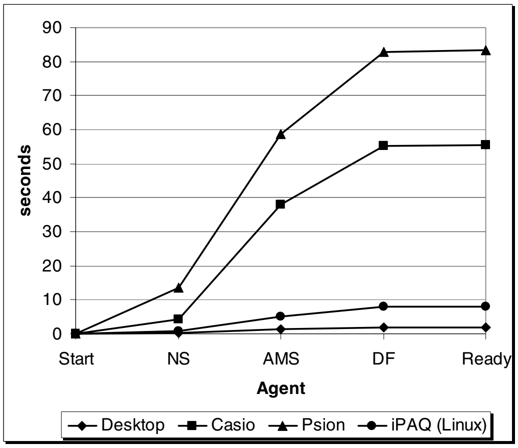


Fig. 2. Results of cumulative platform startup times. Values are averages of five test runs

4.3 Agent Communication

In agent communication tests we measure the time for interaction between three agents. First the requesting agent (Agent 1) finds Agent 2 by querying DF. The DF request message in this case is as simple as it can be; we only specified the service type in the service descriptions. For more information about DF request and service description, see [8]. After Agent 1 has gained knowledge about Agent 2, it sends a ping to Agent 2 according to the FIPA-Request interaction protocol [11] asking Agent 2 to respond to the ping. The sequence of messages is shown in Figure 3. Agent 2 sends back an agree message indicating that is has agreed to perform the request. Thereafter, an inform (pong) is sent back to Agent 1.

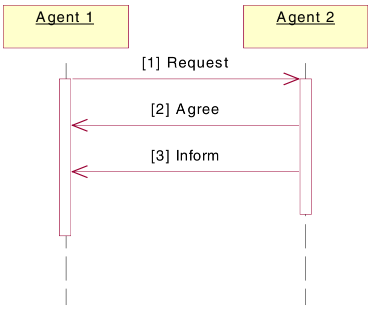


Fig. 3. FIPA-Request interaction protocol

In Figure 4 the results of the test case as seen from Agent 1's perspective are shown. It must be noted that as in Figure 2, here the times presented are cumulative. The measurements are started as Agent 1 sends the DFSearch request. At "DFSearch" Agent 1 gets back the result from DF. It then constructs the ping-request and sends it to Agent 2 at "Request". Agent 2 immediately processes the request and sends back an agree. Agent 1 receives the agree at "Agree". Agent 2 also immediately constructs the inform message and sends it after the agree. Agent 1 receives this at "Inform".

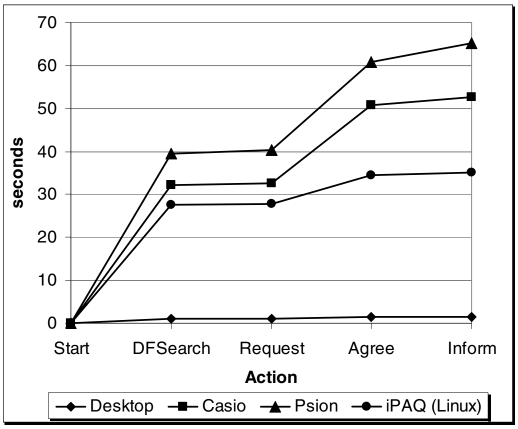


Fig. 4. Results on cumulative communication times. Values are averages of five test runs

It can be seen in the Figure 4, the DFSearch request seems to take almost two times longer to process than the simple ping-pong interaction. This indicates that the processing time of the DFSearch request at the DF takes quite a lot of time. With iPAQ the difference is most significant; DFSearch takes almost 30 seconds to complete, whereas simple ping-pong interaction is completed in few seconds.

4.4 Agent Startup Task Breakdown

In the third test case we inspect a startup of a single agent. The agent startup consists of several tasks[3], which are executed in the following order:

1. Reading the platform profile.
   - Parsing the RDF profile.
   - Constructing and initializing the Profile object.
2. Reading the agent profile.
   - Parsing the RDF profile.
   - Constructing and initializing the Profile object.

[3] For more information about the startup procedure and the internal structure of FIPA-OS agent, see FIPA-OS Developer's Guide [6].

3. Initializing the (persistent) database for the agent.
4. Setting up MTS.
   – Setting up internal MTPs.
   – Setting up possible external MTPs.
   – Creating and initializing the service stack.
5. Setting up the task manager.
6. Setting up the conversation manager.
7. Registering with AMS.
8. Registering with DF.

Instead of just looking at the startup times of each task, we wanted to see how much time relative to the total startup time is spent on each task. In Figure 5 the most time-consuming tasks are depicted. As can be seen in Figure 5, FIPA-OS on the desktop spends most of the time in setting up the MTS while in the small-footprint devices the time portions are more evenly distributed. It is interesting to see that while the desktop computer and iPAQ spend about 30% with the profiles, Casio and Psion devices spend more than half of the total time with them. While Casio processes the profiles almost 60% of the total time, Psion processes them almost 80% of the total time.

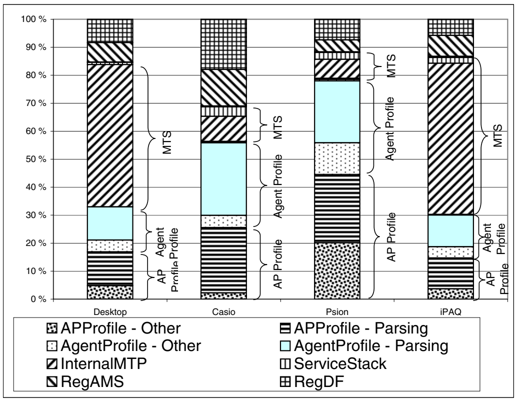


Fig. 5. Subtasks an their relative portion of the total time in agent creation. Results show the averages of five test runs

4.5 XML Parsing

Because XML parsing seems to take most of the time on Casio and Psion devices, we test the pure XML parsing using Xerces 1.3.0, which is used in FIPA-OS. Xerces supports the SAX API [14], which is based on callback events and is inherently faster than DOM [30], which builds a navigable tree representation of XML.

Tests were made using three different, non-validating XML files: `personal.xml`, `rich_iii.xml` and the `platform.profile` of FIPA-OS. The `personal.xml` is a simple XML file containing 37 elements and a total of 242 characters. `rich_iii.xml` is a larger file containing 6631 elements and a total of 179369 characters. The `platform.profile` has only seven elements and 1078 characters. The `personal.xml` file is from the Xerces distribution, `rich_iii` comes with the Crimson parser [14] and the `platform.profile` is a part of FIPA-OS distribution available at the FIPA-OS homepage [16].

Table 2 shows the results of parsing tests. For each item the mean value and standard deviation for five test runs is calculated. All the times are in milliseconds. The three XML files are in their own block.

Table 2. XML parser results.

	Desktop		**Casio**		**Psion**	
	μ	σ	μ	σ	μ	σ
			`personal.xml`			
create parser	63	1,9	2125	30,4	2410	6,9
1st parsing	132	5,5	5436	46,2	5825	7,2
class loading	121	5,8	5037	46,5	5428	14,0
2nd parsing	11	0,5	408	9,3	466	7,1
3rd parsing	10	0,4	399	10,1	391	0,0
renew parser	2	0,0	118	2,4	88	8,8
new parsing	20	4,8	441	3,9	234	0,5
			`rich_iii.xml`			
create parser	62	0,5	2163	20,0	2412	8,8
1st parsing	933	16,3	27284	344,3	23619	8,5
class loading	121	24,7	3036	90,5	5262	33,8
2nd parsing	804	4,4	24590	474,9	18563	0,5
3rd parsing	812	14,7	24248	322,2	18356	28,3
renew parser	10	0,0	262	6,4	94	0,0
new parsing	804	4,7	24183	387,3	18234	0,4
			`platform.profile`			
create parser	64	2,3	2073	15,0	2403	6,7
1st parsing	102	6,3	3248	254,1	4569	9,1
class loading	91	6,3	2822	238,9	4147	9,1
2nd parsing	12	1,2	437	16,3	431	8,4
3rd parsing	11	0,0	426	15,5	422	0,0
renew parser	2	0,0	121	3,2	87	8,0
new parsing	20	1,3	465	16,8	459	8,8

“Create parser” is the time for creating the parser object. “1st parsing” corresponds to the time for parsing the original file for the first time. It must be noted that this also includes the time needed for all the parsing-related Java classes being loaded into the memory. “Class loading” is only the time which is needed for the parsing related classes to be loaded into the memory. “2nd parsing” is the time needed for the second parsing of the original file as in the 1st parsing. “3rd parsing” is the time for the third parsing of the original file, except that between the 2nd and 3rd parsing, a different file was parsed. “Renew parser” is the time for nullifying the parser and creating another one. “New parsing” corresponds to the time needed for the 1st parsing of the original file with the renewed parser object.

Firstly, the performance compared to a desktop computer is poor. Figure 6 depicts the performance measures of `platform.profile`. We can see that the parsing in Casio took 32 times and in Psion 45 times longer than in the desktop computer.

Secondly, when comparing the parsing times of the first and second parsings, it can be seen that the class loading takes a huge amount of time compared to the actual parsing, even if the parsed file changes. Whereas the first parsing on Casio takes over three seconds, the subsequent ones take only less than half a second. Therefore, when optimizing parsing, it must be assured that the class loading is done only once and the parser classes are reused as much as possible in terms of free memory.

Thirdly, Xerces is unsuitable for constrained environments because of its size of about 1.6 MB. For small-footprint devices, a more suitable choice as XML parser could be a minimal XML parser [5], however XML documents may need to be minimized first. Another choice could be a smaller, non-validating parser such as Crimson [14], which requires only 200KB of storage space.

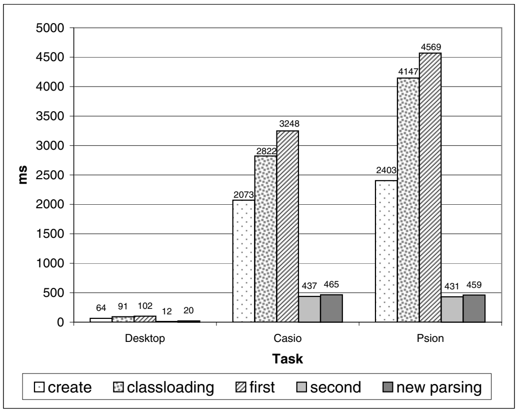


Fig. 6. XML parser performance with `platform.profile`

5 Discussion

Issues such as the implementation of the Java virtual machine, the operating system, the number of other resident programs and Java's unpredictable garbage collection make the analysis of multithreaded systems difficult. However, as the current and future PDAs are gaining more computing power and functionality, it becomes possible to execute Java-based software agents on mobile devices for increased interoperability and portability. However, large frameworks designed to be used in traditional computing environments do not necessarily scale well to small devices. We have observed that the execution behaviour may differ between different hardware platforms and issues such as parsing, object creation and data communication may need to be modified.

In our opinion when dealing with small-footprint devices the traditional client-server architecture should be replaced with client-mediator-server architecture, where the heaviest operations of the client side are executed in the mediator. Figure 7 shows how FIPA-OS could be partitioned to achieve this. Mobile device only runs a light-weight version of FIPA-OS, whereas the access node would act as a proxy for the mobile executing functions that are heavy for the mobile device. Mobile device and access node could then, because logically being one FIPA-OS platform, use some communication scheme especially designed for wireless environments.

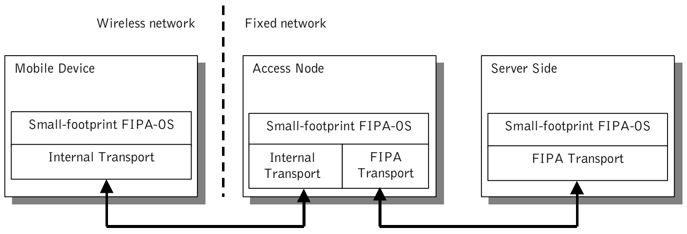


Fig. 7. Partitioning FIPA-OS between mobile device and access node

As the overall performance of the current small-footprint devices's Java environment is poor, it is hard to give solid instructions on how to optimize FIPA-OS for small-footprint devices. Based on the test, we propose the following optimizations for reducing functionality and complexity:

- Replace RMI with direct method calls
- Remove/simplify static XML profile parsing
- Replace runtime XML parser with more lightweight one
- Move heavy computations to the server side (access node)
- Lighten threading and apply thread pooling
- Make lightweight versions of MTS, Conversation Manager and Task Manager
- Apply pure Java optimizations

With the help of various profiles that define pluggable components, it is possible to create an adaptable and scalable agency that can be used in a wide range of different environments. Also, especially the profile parsing could be optimized by reusing the parser classes as much as possible.

6 Conclusion and Future Work

As can be observed in the test results, running non-optimized Java applications on small-footprint devices is not feasible, because the devices are still too limited in terms of CPU power and the amount of memory. For instance, although FIPA-OS runs on the devices, it runs too slowly to be usable for real applications. Thus, FIPA-OS as such is not applicable for small-footprint devices, but could be optimized for them.

Based on our test results, the most important optimizations are: replacing RMI with direct method calls, reducing the amount of threading and optimizing parsing. As a conclusion, with the currently available small-footprint devices, software has to be simple for feasible performance.

The CRUMPET project [20] addresses these issues with one goal being to design and implement a MicroFIPA-OS agent platform suitable for small-footprint devices. The implementation will be ready in the beginning of year 2002.

References

1. Infrared Data Association. IrDA homepage, 2001. Available at http://www.irda.org.
2. Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE - A FIPA-compliant agent framework. In *Proceedings of the 4th International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-99)*, pages 97–108, 1999.
3. Spell Brett. *Professional Java Programming.* Wrox Press Ltd, Birmingham, UK, 2000.
4. Pendragon Software Corporation. CaffeineMark 3.0 homepage, 2001. Available at http://www.pendragon-software.com/pendragon/cm3/.
5. Docuverse. Min - Minimal XML Parser homepage, 2001. Available at http://www.docuverse.com/min/.
6. Emorphia Ltd. *FIPA-OS Developer's Guide*, February 2001. Available at http://fipa-os.sourceforge.net/docs/Developers_Guide.pdf.
7. Joshua Engel. *Programming for the Java Virtual Machine.* Addison-Wesley Longman, Inc., Reading, Massachusetts 01867, June 1999.
8. Foundation for Intelligent Physical Agents. *FIPA Abstract Architecture Specification.* Geneva, Switzerland, November 2000. Work in progress. Specification number PC00001, available at http://www.fipa.org/specs/fipa00001/.
9. Foundation for Intelligent Physical Agents. *FIPA ACL Message Structure Specification.* Geneva, Switzerland, October 2000. Work in progress. Specification number XC00061, available at http://www.fipa.org/specs/fipa00061/.
10. Foundation for Intelligent Physical Agents. *FIPA Communicative Act Library Specification.* Geneva, Switzerland, November 2000. Work in progress. Specification number PC00037, available at http://www.fipa.org/specs/fipa00037/.

11. Foundation for Intelligent Physical Agents. *FIPA Request Interaction Protocol Specification*. Geneva, Switzerland, October 2000. Work in progress. Specification number PC00026, available at http://www.fipa.org/specs/fipa00026/.
12. Foundation for Intelligent Physical Agents. *FIPA SL Content Language Specification*. Geneva, Switzerland, November 2000. Work in progress. Specification number XC00008, available at http://www.fipa.org/specs/fipa00008/.
13. Foundation for Intelligent Physical Agents. FIPA Homepage, 2001. Available at http://www.fipa.org/.
14. The Apache Software Foundation. Apache XML Project Homepage, 2001. Available at http://xml.apache.org/.
15. Nick Jennings and Michael Wooldridge. *Agent Technology: Foundations, Applications, and Markets*. Springer, Berlin Heidelberg, Germany, 1998.
16. Emorphia Ltd. FIPA-OS homepage, 2001. Available at http://sourceforge.net/projects/fipa-os/.
17. Metamata, Inc. JavaCC - The Java Parser Generator, 2001. Available at http://www.metamata.com/javacc/index.html.
18. Jennings N., Sycara K., and Wooldridge M. *A Roadmap of Agent Research and Development*, pages 275–306. Kluwer Academic Publisher, Boston, 1998.
19. S. Poslad, P. Buckle, and R. Hadingham. FIPA-OS: the FIPA agent Platform available as Open Source. In Jeffrey Bradshaw and Geoff Arnold, editors, *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, pages 355–368, Manchester, UK, April 2000. The Practical Application Company Ltd.
20. The CRUMPET Project. CRUMPET Project homepage, 2001. Available at http://www.ist-crumpet.org.
21. Kravets R. and Krishnan P. Application-driven Power Management for Mobile Communication. *Wireless Networks*, 6(4):263–277, 2000.
22. Bagrodia Rajive, Chu Wesley W., Kleinrock Leonard, and Popek Gerald. Vision, Issues, and Architecture for Nomadic Computing. *IEEE Personal Communications*, 1995.
23. Campadello S., Helin H., Koskimies O., and Raatikainen K. Wireless Java RMI. In *Proceedings of The 4th International Enterprise Distributed Object Computing Conference*, pages 114–123, September 2000.
24. Hirano S. HORB Homepage, 2000. Available at http://www.horb.org/horb/.
25. Sun Microsystems, Inc. *Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices*, June 2000. Available at http://java.sun.com/products/cldc/wp/KVMwp.pdf.
26. Sun Microsystems, Inc. J2ME homepage, 2001. Available at http://java.sun.com/j2me/.
27. Sun Microsystems, Inc. PersonalJava (TM) Application Environment homepage, 2001. Available at http://java.sun.com/products/personaljava/.
28. Symbian. The Symbian homepage, 2001. Available at http://www.symbian.com.
29. Kamba T., Elson S. A., Harpold T., Stamper T., and Sukaviriya P. Using Small Screen Space More Efficiently, 1996.
30. W3C. *Document Object Model (DOM) Level 2 Core Specification, version 1.0*, November 2000. Available at http://www.w3.org/TR/DOM-Level-2-Core/.

Author Index