Lecture Notes in Artificial Intelligence 2977

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Giovanna Di Marzo Serugendo
Anthony Karageorgos   Omer F. Rana
Franco Zambonelli (Eds.)

Engineering Self-Organising Systems

Nature-Inspired Approaches to Software Engineering

Springer

Volume Editors

Giovanna Di Marzo Serugendo
University of Geneva
Centre Universitaire d'Informatique, 1211 Geneva 4, Switzerland
E-mail: Giovanna.Dimarzo@cui.unige.ch

Anthony Karageorgos
University of Thessaly
Department of Computer and Communication Engineering
37 Glavani, 28th October Str., Deligiorgi Building
4th floor, room D3/4, 382 21 Volos, Greece
E-mail: karageorgos@computer.org

Omer F. Rana
Cardiff University
School of Computer Science
5 The Parade, P.O.Box 916, Cardiff CF24 3XF, UK
E-mail: o.f.rana@cs.cardiff.ac.uk

Franco Zambonelli
Università di Modena e Reggio Emilia
Dipartimento di Scienze e Metodi dell'Ingegneria
42100 Reggio Emilia, Italy
E-mail: franco.zambonelli@unimore.it

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): I.2.11, I.2, C.2.4, F.1.1, D.2

ISSN 0302-9743
ISBN 3-540-21201-9 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

© Springer-Verlag Berlin Heidelberg 2004
Printed in Germany

Typesetting: Camera-ready by author, data conversion by DA-TeX Gerd Blumenstein
Printed on acid-free paper SPIN: 10989791 06/3142 5 4 3 2 1 0

Self-Organisation: Paradigms and Applications

Giovanna Di Marzo Serugendo, Noria Foukia, Salima Hassas,
Anthony Karageorgos, Soraya Kouadri Mostéfaoui, Omer F. Rana,
Mihaela Ulieru, Paul Valckenaers, and Chris Van Aart

Engineering Self-Organising Applications Working Group
http://www.agentcities.org/Activities/WG/ESOA/

Abstract. A self-organising system functions without central control, and through contextual local interactions. Components achieve a simple task individually, but a complex collective behaviour emerges from their mutual interactions. Such a system modifies its structure and functionality to adapt to changes to requirements and to the environment based on previous experience. Nature provides examples of self-organisation, such as ants food foraging, molecules formation, or antibodies detection. Similarly, current software applications are driven by social interactions (negotiations, transactions), based on autonomous entities or agents, and run in highly dynamic environments. The issue of engineering applications, based on the principles of self-organisation to achieve robustness and adaptability, is gaining increasing interest in the software research community. The aim of this paper is to survey natural and artificial complex systems exhibiting emergent behaviour, and to outline the mechanisms enabling such behaviours.

Keywords: Self-organisation, self-organising application, emergence, collective behaviour, multi-agent systems.

1 Introduction

The study of self-organising systems is a field that has been explored at least since 1953 with the work done by Grassé [25], who studied the behaviour of insect societies. Many systems in nature demonstrate self-organisation, such as planets, cells, organisms and societies. All these systems exhibit recurrent properties inherent to self-organisation. The simplest form of self-organisation can be achieved by the arrangement of parts of a system in such a way as to be non-random. Considerable research has already been undertaken to study such systems. Self-organising systems are often encountered in many scientific areas including biology, chemistry, geology, sociology, and information technology.

A large number of software self-organising systems are designed based on natural mechanisms of self-organisation. Furthermore, recent research has been oriented towards introducing self-organisation mechanisms specifically for software applications, as well as entire development techniques supporting self-organisation [19]. This trend originates from the fact that current software applications need to cope with requirements and constraints stemming from the

© Springer-Verlag Berlin Heidelberg 2004

increased dynamism, sophisticated resource control, autonomy and decentralisation inherent in contemporary business and social environments. The majority of these characteristics and constraints are the same as those which can be observed in natural systems exhibiting self-organisation.

This survey firstly defines self-organisation, and presents examples of self-organising systems taken from natural life. Subsequently, it describes the different mechanisms enabling social, natural and artificial organisations to achieve a coherent global behaviour. Finally, it reviews several software applications exhibiting a self-organising behaviour.

2 Self-Organising Systems

The notion of self-organisation is popular in many different research fields. Therefore, it is difficult to find a precise and concise definition of what is the meaning of the term self-organisation. However, it seems that similar properties are apparent and recurrent among the different definitions and research fields referring to self-organisation. The next sub-section defines what are the inherent characteristics of self-organisation. Subsequently, the following sub-sections describe a number of different types of self-organising systems. It is not the intention of this paper to provide an exhaustive review of all types of self-organising systems. Instead, self-organising systems are classified in three broad categories: physical, living and social systems. For each category a representative example is described.

2.1 Definition

Intuitively, self-organisation refers to the fact that a system's structure or organisation appears without explicit control or constraints from *outside* the system. In other words, the organisation is intrinsic to the self-organising system and results from *internal* constraints or mechanisms, due to *local interactions* between its components [11]. These interactions are often *indirect* thanks to the environment [25]. The system dynamics modifies also its *environment*, and the modifications of the external environment influence in turn the system, but without disturbing the internal mechanisms leading to organisation. The system evolves *dynamically* [9] either in time or space, it can maintain a stable form or can show transient phenomena. In fact, from these interactions, *emergent* properties appear transcending the properties of all the individual sub-units of the system [30]. One well-known example is that of a colony of ants sorting eggs without having a particular ant knowing and applying the sorting algorithm. The emergence is the fact that a structure, not explicitly represented at a lower level, appears at a higher level. With *no central control*, a complex collective behaviour raises then from simple local individual interactions. More generally, the field of complex systems studies emergent phenomenon, and self-organisation [8].

2.2 Physical Systems

A characteristic of physical self-organising systems is that many of them present a so called *critical value* in which the state of the system changes suddenly to another state under certain conditions (temperature, pressure, speed, ...). Thus, self-organisation is observed *globally* when the physical system moves from a chaotic disordered state to a stable one. For instance, a thermodynamic system such as a gas of particles has emergent properties, temperature and pressure, that do not derive from the description of an individual particle, defined by its position and velocity. Similarly, chemical reactions create new molecules that have properties that none of the atoms exhibit before the reaction takes place [8]. Moreover, the magnetisation of a multitude of spins[1] is a clear case of self-organisation because, under a certain temperature, the magnetic spins spontaneously rearrange themselves pointing all in the same direction thanks to a strong emerging magnetic field.

2.3 Living Systems

A scientific aspect in the study of living organisms is the determination of invariant in the evolution of natural systems. In particular the spontaneous appearance of an order in living complex systems due to the self-organisation. In the optic of biological research, the global emergence of a behaviour or a feature that can not be reduced to the properties of each system's component (molecules, agents, cells, ...) defines also the common meaning of self-organisation. One example described in [43] is the self-organisation of the cytoskeleton thanks to collective processes of reaction and diffusion of the cytoskeletal filaments. The cytoskeleton is the basis of the internal architecture of the cytoplasm of eukaryotic cells. Eukaryotic cells are cells of the higher organisms, containing a true nucleus bounded by a nuclear membrane. These cells are founded in animals and plants. Eukaryotic cells are often organised into organs to create higher levels of complexity and function thanks to metabolic processes. The obtained organ has a defined functionality that transcends all the functionality offered by each of its constitutive cells. These cells are the basic functioning units of life and evolve in step through external changes with the environment. These units of life (cells) have to use internal metabolic processes such as mutation or natural selection, to adapt to natural life's evolution. This is also the result of the self-organisation of cells. Other examples in human body are the human nervous system, or the immune system. Such living systems transparently manage vital functions, such as blood pressure, digestion, or antibodies creation.

2.4 Social Systems

Social insects organise themselves to perform activities such as food foraging or nests building. Cooperation among insects is realised through an indirect

[1] A spin is a tiny magnet.

communication mechanism, called stigmergy, and by interacting through their environment. Insects, such as ants, termites, or bees, mark their environment using a chemical volatile substance, called the pheromone, for example, as do ants to mark a food trail. Insects have simple behaviour, and none of them alone "knows" how to find food but their interaction gives rise to an organised society able to explore their environment, find food and efficiently inform the rest of the colony. The pheromonal information deposited by insects constitutes an indirect communication means through their environment.

Apart from animal societies, human beings organise themselves into advanced societies. Human beings use direct communication, they engage in negotiation, build whole economies and organise stock markets.

3 Self-Organising Mechanisms

This section presents the major self-organising mechanisms used in natural and software systems to achieve self-organisation.

3.1 Magnetic Fields

A self-organisation phenomenon has been studied in the structure of a piece of potentially magnetic material. A magnetic material consists of a multitude of tiny magnets or spins. The spins point in different directions cancelling their respective magnetic fields. At a lower level, the orientation of the spins is due to the random movements of the molecules in the material: the higher the temperature, the stronger these random movements of the molecules in the material. These molecular movements affect the spins making them difficult to orient in an ordered way. However, if the temperature decreases the spins spontaneously point in the same direction. In this case, the different magnetic fields now add up, producing a strong overall field. Magnetisation exhibits self-organisation because the orientation of the spins is variable and depends on the local neighbourhood. Under low temperature, the force between neighbouring spins is dominating and they tend to build order. Similar phenomena, are observed in the crystallisation from a liquid state, which is another common example of self-organisation.

3.2 Kohonen Neural Networks

Kohonen neural networks, also called self-organising maps, are useful for clustering applications [36]. They take their inspiration from brain cells, which are activated depending on the subject's location. Such a network is made of two neurons layers (input, and output), and usually follows a regular two-dimensional grid of neurons. This grid represents a topological model of the application to cluster. Indeed, the network maps similar data to the same, or adjacent, node of the grid, by projecting multi-dimensional vectors of data onto a two-dimensional regular grid, preserving the data clustering structure.

To each neuron is associated a weight vector, randomly initialised. This topology preserving behaviour of the network is obtained through a learning rule which determines the *winner* among the neurons, as being the one whose weight vector is closer to the vector of the sampled data entered in the network. Data will then be affected to this neuron. Once the winner is determined, an update of the weight vectors of each neuron is performed, in order to reinforce clustering. On the basis of this algorithm, a method, called WEBSOM [37], has been defined, which helps organise heterogeneous text documents onto significant maps.

3.3 Stigmergy

Social insect societies (ants, bees, wasps, termites, etc) exhibit many interesting complex behaviours, such as emergent properties from local interactions between elementary behaviours achieved at an individual level. The emergent collective behaviour is the outcome of a process of self-organisation, in which insects are engaged through their repeated actions and interactions with their evolving environment [30]. Self-organisation in social insects relies on an underlying mechanism, the mechanism of stigmergy, first introduced by Grassé in 1959 [25]. Grassé studied the behaviour of a kind of termites during the construction of their nests and noticed that the behaviour of workers during the construction process is influenced by the structure of the constructions themselves. This mechanism is a powerful principle of cooperation in insect societies. It has been observed within many insect societies like those of wasps, bees and ants. It is based on the use of the environment as a medium of inscription of past behaviours effects, to influence the future ones. This mechanism defines what is called a self-catalytic process, that is the more a process occurs, the more it has chance to occur in the future. More generally, this mechanism shows how simple systems can produce a wide range of more complex coordinated behaviours, simply by exploiting the influence of the environment. Much behaviour in social insects, such as foraging or collective clustering are rooted on the stigmergy mechanism. Foraging is the collective behaviour through which ants collect food by exploring their environment. During the foraging process, ants leave their nest and explore their environment following a random path. When an ant finds a source of food, it carries a piece of it and returns back to the nest, by laying a trail of a hormone called pheromone along its route. This chemical substance persists in the environment for a particular amount of time before it evaporates. When other ants encounter a trail of pheromone, while exploring the environment, they are influenced to follow the trail until the food source, and enforce in their coming back to the nest the initial trail by depositing additional amounts of pheromone. The more a trail is followed, the more it is enforced and has a chance of being followed by other ants in the future. Collective Sorting is a collective behaviour through which some social insects sort eggs, larvae and cocoons [17]. As mentioned in [10], an ordering phenomenon is observed in some species of ants when bodies of dead ants are spread in the foraging environment. Ants pick up dead bodies and drop them later in some area. The probability of picking up an item is correlated with the density of items in the region where the operation occurs. This behaviour has been studied in robotics through simulations [29] and real implementations [30]. Robots with primitive behaviour are able to achieve a spatial environment structuring, by forming clusters of similar objects via the mechanism of stigmergy described above.

Tag-Based Models. In addition to the digital pheromone, which is the artificial counterpart of the natural pheromone used by the ants, new electronic mechanisms directly adapted to software applications are being developed. The notion of tags, a mechanism from simulation models, is one of them. Tags are markings attached to each entity composing the self-organising application [26]. These markings comprise certain information on the entity, for example functionality and behaviour, and are observed by the other entities. In this case the interaction would occur on the basis of the observed tag. This would be particularly useful to let interact electronic mobile devices that do not know each other in advance. Whenever they enter the same space, for example a space where they can detect each other and observe the tags, they can decide on whether they can or cannot interact.

3.4 Coordination

The social aspect of multi-agent systems is engineered through coordination models that define the agent interactions in terms of interaction protocols and interaction rules. In other words, a coordination model defines how agents interact and how their interactions can be controlled [15]. This includes dynamic creation and destruction of agents, control of communication flows among agents, control of spatial distribution and mobility of agents, as well as synchronisation and distribution of actions over time [12]. In general, a coordination model is defined by: (a) *coordinable entities* (components), these are the agents, which are coordinated. Ideally, these are the building blocks of a coordination architecture, for example agents, processes, tuples, atoms, etc.; (b) *coordination media*, these are the coordinators of inter-agent entities. They also serve to aggregate a set of agents to form a configuration, for example channels, shared variables, tuple spaces; and (c) *coordination laws* ruling actions by coordinable entities or the coordination media. The laws usually define the semantics of a number of coordination mechanisms that can be added to a host language [16].

A particular coordination model depends on the coordination media, coordination laws, and the programming language used for expressing coordinables. In control-driven models agents interact either by broadcasting events to the other agents, or through a point-to-point channel connection. Communication among the agents is established by a third party coordinator process. Coordination laws establish propagation of events, dynamic reconnection of ports, and creation and activation of processes.

In data-driven models, the coordination media is a shared data space, and interactions consists in asynchronously exchanging data through the data space.

Coordination laws govern data format and primitives for storing, removing, and retrieving them from the interaction space. Such models are derived from Linda [24], an early coordination model based on shared tuple spaces. This model fits well with multi-agent systems because it allows interactions among anonymous entities, joining and leaving the system continuously, in a not-predefined manner. In order to incorporate more control over interactions, into data-driven models, hybrid models are also considered. In such models, the data space serves for responding to communication events. It becomes a programmable coordination medium where new coordination primitives and new behaviours can be added in response to communication events. Acting as middleware layers, coordination spaces provide uncoupled interaction mechanisms among autonomous entities, which input data into a common tuple space, and may retrieve data provided by other entities. These models support a limited form of self-organisation, since they enable decentralised control, anonymous and indirect local interactions among agents.

Tuples on the Air. On top of the basic coordination environment, several enhancements have been realised in order to address specifically self-organisation. The TOTA environment (Tuples On The Air) propagates tuples, according to a propagation rule, expressing the scope of propagation, and possible content change [39]. If we come back to the metaphor of ant societies, such a model allows, among others, to electronically capture the notion of digital pheromone, deposited in the tuple space and retrieved by other agents. The propagation rule removes the pheromone from the data space, once the evaporation time has elapsed.

Coordination Fields. Alternatively, the Co-Fields (coordination fields) model drives agents behaviour as would do abstract force fields [40]. The environment is represented by fields, which vehicle coordination information. Agents and their environment create and spread such fields in the environment. A field is a data structure composed of a value (magnitude of field), and a propagation rule. An agent then moves by following the coordination field, which is the combination of all fields perceived by the agent. The environment updates the field according to the moves of the agents. These moves modify the fields which in turn modify the agents behaviour. This model allows representing not only complex movements of ants, and birds, but also tasks division and succession.

4 Self-Organising Applications

Nature provides examples of emergence and self-organisation. Current distributed applications, as well as applications of a near future, already show a self-organising behaviour, since they are situated in highly changing environments, they are made of a large number of heterogeneous components and cannot undergo a central control.

4.1 Multi-agent Systems

An agent is a physical (robot), or a virtual (software) entity situated in an environment that changes over time: the physical world or an operating system respectively. Through its sensors, the agent is capable of perceiving its environment, and through its effectors, it is capable of performing actions that affect the environment. For instance, a robot may take notice of obstacles with an embedded camera, and to remove them from its way with an articulated arm. A software agent may understand a user's request through a user's interface, and send an e-mail to the user once the request has been satisfied [46, 32].

Every single agent has one or more limitations, which can be categorised into cognitive limitations, physical limitations, temporal limitations and institutional limitations. Cognitive limitations resemble the fact that individuals are rationally bounded. It means that the data, information, and knowledge an individual can process and the detail of control an individual can handle is limited. As tasks grow larger and more complex, techniques must be applied to limit the increase of information and the complexity of control. Individuals can be limited physically, because of their physiology or because of the resources available to them. Temporal limitations exist where the achievement of individual goals exceeds the lifetime of an individual, or the time over which resources are available for achieving a goal. Finally, individuals can be legally or politically limited.

To overcome their limitations, agents group together and form *multi-agent systems*, or societies of agents, where they work together to solve problems that are beyond their individual capabilities [44]. A robot has to bring a cup from the office to the kitchen. It may ask the need of another agent to bring the cup in the kitchen, if it is itself unable to reach the kitchen whose door is closed. On the basis of its knowledge about the user's behaviour, an assistant agent may decide to regularly inform that user about new computer science books, without the user having explicitly notified the agent to do that. In order to obtain this information, the agent may need to contact other agents aware of computer science books.

Agents interact (communicate, coordinate, negotiate) with each other, and with their environment. Interactions among agents usually follow a coordination model as explained in Subsection 3.4. An agent can communicate directly or indirectly with other agents for cooperation or competition purposes. Since the agent perceives its environment and interacts with other agents, it is able to build a partial representation of its environment, which constitutes its knowledge. Usually, in a multi-agent system, interaction are not pre-defined, and there is no global system goal. The interaction dynamics between an agent and its environment lead to emergent structure or emergent functionality, even though no component is responsible for producing a global goal.

4.2 Grid

Computational Grids provide the software and networking infrastructure required to integrate computational engines/scientific instruments, data repositories, and human expertise to solve a single large problem (generally in science and engineering domains). Computational engines can comprise of specialist, tightly coupled architectures (such as parallel machines) or loosely coupled clusters of workstations. There has been an emerging interest in trying to integrate resources across organisational boundaries through file or CPU sharing software (such as KaZaA [4] and Gnutella [3] for file sharing and Entropia [2] and UD [5] for CPU sharing). These individual resources are often geographically distributed, and may be owned by different administrators (or exist within different independently administered domains). Managing resources within Computational Grids is currently based on infrastructure with centralised registry and information services (based on the LDAP/X500 directory service) - such as provided by the Open Grid Services Infrastructure (OGSI) [1]. In this process, resource owners must register their capabilities with a limited number of index servers, enabling subsequent search on these servers by resource users. The provision of such centralised servers is clearly very limiting, and restricts the scalability of such approaches. Current resources being provided within Computational Grids are owned by national or regional centres (or by research institutions), and therefore concerns regarding access rights and usage need to be pre-defined and approved. However, as resources from less trusted users are provided, the need to organise these into dynamic communities, based on a number of different criteria: performance, trust, cost of ownership and usage, usability, etc. become significant. Self-organisation therefore plays an important role in identifying how such communities may be formed and subsequently dis-banded. A utility-based approach for forming such communities is explored in [38]. However, it is necessary to understand and investigate alternative incentive structures that will enable the formation of such communities.

4.3 Service Emergence

Itao et al. [31] propose Jack-in-the-Net (Ja-Net), a biologically-inspired approach to design emergent network applications and services in large-scale networks. In Ja-Net, network applications and services are dynamically created from local interactions and collaboration of self-organising entities, called cyber-entities. Each cyber-entity implements a simple functional component of the overall service or application. Furthermore, it follows simple behavioural rules, similar to the behaviours of biological entities, such as: energy exchange with the environment, migration or replication, or relationship establishment. Services emerge from strong relationships among cyber-entities. Indeed, cyber-entities record information about peer cyber-entities during a relationship. Once relationships among a collection of cyber-entities are strong enough, they form a group and create a service. Relationship strengths are evaluated on the basis of the utility degree of each cyber-entity participating in the service. The utility degree is estimated using user's feedback on the delivered service. In addition to service emergence, the system exhibits a natural selection mechanism based on the notions of energy stored or spent by cyber-entities and the diversity of services created. Due to the migration, replication and possible deaths of entities, the system is also able to adapt to networks changes.

4.4 Web Communities

Flake et al. [21] have shown using a specific algorithm that Web pages form related communities. A Web page is a member of a community if it has more hyperlinks within the community than outside it. At the human level, one cannot have an overall picture of the structure, based on hyperlinks, that emerges among the Web pages. Flake et al. have defined a specific algorithm which highlights communities of related Web pages, on the basis of hyperlinks contained in the pages.

From a self-organisation point of view, authors of Web pages simply put them on the Web with hyperlinks on other pages. Inserting a page on the Web modifies the environment, for example the world space of the Web pages, and this in turn modifies the behaviour of other authors of pages. Indeed, it is now possible to reach existing pages from a new location and it is possible to reference and go through these new pages. By analogy with the ants metaphor, authors place Web pages (pheromone) on the Web (food trail). These Web pages contain specific information for other authors, who will reinforce (or not) the strengths among Web pages by referencing them. Authors then collectively but independently organise Web pages into communities.

4.5 Networking with Ants

Di Caro et al. [18] suggests using artificial ant-based modelling to solve network problems. The motivation is that ants modelling might be able to cope with communication networks better than humans. A first survey [9], dealing with several swarm intelligence examples in social insect communities, shows how ants-like behaviour (ants, bees, termites, and wasps) provide a powerful metaphor to build a completely decentralised system. Such a system is composed of individual and simple entities, which collaborate to allow a more complex and collective behaviour. The global emergent behaviour of the ant population is due to a network of interactions between the ants themselves, but also between the ants and their environment. This emergent collective behaviour allows the social insect colony to organise vital tasks such as finding food, building the nest, dividing labour among individuals and spreading alarm among members of the society. Many of these tasks and their respective mechanism have inspired computer network scientists notably to mimic ant foraging behaviour to optimise the routing in communication networks or to mimic the division of labour and the task allocation to optimise the load balancing in network systems. When ants forage, they wander randomly starting from their source (nest) until they reach their destination (food) and on their way they lay a chemical trail termed *pheromone*. The pheromone deposited along the path followed by each ant marks this path for other ants. In fact, the more one path is marked the more it will be chosen by other ants. This mechanism where the environment becomes the communication medium is termed *stigmergy* (see Subsection 3.3). To apply this paradigm to network routing, Dorigo and his colleagues built an artificial ant network where periodically each node launches an ant to find the route to a given destination.

By simply smelling the strength of the pheromones along the neighbourhood paths of the node, the ant generates the map that shows the fastest route to any end point. In case of congestion, it was showed that this mechanism outperforms all other popular routing algorithms in terms of speed achieved to avoid the traffic jams.

4.6 Network Security

The use of Mobile Agents in sophisticated applications offers advantages for constructing flexible and adaptable wide-area distributed systems. Notably, applications such as Intrusion Detection Systems (IDSs) and Intrusion Response Systems (IRSs) have become even more relevant in the context of large-scale network infrastructures, where traditional security mechanisms demonstrate severe weaknesses [41]. Indeed, as they can be retracted, dispatched, cloned or put in stand-by, mobile agents have the ability to sense network conditions and to load dynamically new functionality into a remote network node (such as a router). The network awareness of mobile agents can also significantly contribute to the detection of intrusions and enables providing appropriate response. Deriving from this trend, mobile agents have been proposed as support for Intrusion Detection (ID) and Intrusion Response (IR) in computer networks. The originality of this approach lies on the design of the intrusion detection and response system (IDRS). Indeed, the organisation of mobile agents follows the behaviour of natural systems to detect an intrusion as well as to answer an intrusion [23]. Schematically there are two natural paradigms that have been referred to. Firstly, the *human immune system*, because the IDS is based upon principles derived from the immune system model, where Intrusion Detection Agents (ID Agents) map the functionality of the natural immune system to distinguish between normal and abnormal events (respectively ”self” and ”non self” in the immune system) as explained in [22]. Secondly , the social insect *stigmergy* paradigm, because the IRS is based upon principles derived from this paradigm. In fact, Intrusion Response Agents (IR Agents) map the collective behaviour of an ant population by following a synthesised electronic pheromone specific to the detected intrusion until the source of the attack – in order to perform its response task. This pheromone has been previously diffused throughout the network by an ID Agent when it detected the attack. This kind of collective paradigm is very interesting because it consists in having each ant execute a rather light task (mobile agents play the role of ants in the IR System) to induce collectively a more complex behaviour. This approach is also very powerful because the ID System, as well as the IR System, are completely distributed in the network without any centralised control: both systems are essentially constituted by mobile agents which travel across the network, dynamically adjusting their routes according to collected events, without any simple way to trace them. Furthermore, mobile agents are quite polyvalent because they can detect and respond to intrusion. This enhances the difficulty for an attacker to distinguish between ID Agents and IR Agents.

4.7 Robots

Researchers have also been inspired by living systems to build robots. Lots of recent researches in robotics use insect-based technology where robots self-organise to accomplish a task (gathering a set of objects at a precise location for instance). As it is the case for ants' populations, the populations of robots own a local view of their environment, they can share individual information with other robots and co-operate with them. One direct application of the self-organisation with robots is the building of a global cartography of the environment where they are immersed without having each the knowledge of the global topology of the environment. In the approach described in [7] the robots' perception of their environment is tightly connected to their action, similarly to many successful biological systems. Robots perceive their environment locally by sending a simple visual information to their control system. The overall behaviour of the system emerges from the coordination, integration and competition between these various visual behaviours.

4.8 Manufacturing Control

The food foraging behaviour in ant colonies has been translated into a design for agent societies performing manufacturing control. Resource agents provide a reflection of the underlying production system in the world of the agents. These resource agents offer a space for the other agents to navigate through - each agent knowing its neighbours - and offer spaces on which information can be put, observed and modified - like the ants leave pheromones in the physical world. Virtual agents - ants - move through this reflection of the physical world and collect information, which they make available elsewhere. Firstly, these ants collect information about the available processes, travel upstream and place the collected information at routing points. Secondly, ants explore possible routings for the products being made, make a selection and propagate the resulting intentions through the 'reflection'. Resource agents receive this information about the intentions of their users and compile short-term forecasts for themselves. These forecasts allow up-to-date predictions of processing times used by the ants exploring routes and propagating intentions. All these activities are subjected to an evaporation (time-out) and refresh process that enables the system to keep functioning in a highly dynamic environments (frequent changes and disturbances) [51].

4.9 Self-Organising Sensor Networks

Self-organising wireless sensor networks are used for civil and military applications, such as volcanoes, earthquakes monitoring and chemical pollution checking. Sensor networks consist of self-organised nodes, which dynamically need to set up an ad-hoc P2P network, once they are deployed in a given area. They need as well to calibrate themselves in order to adapt to their environment [54].

Sensor networks benefit also of recent technology enabling integration of a complete sensor system into small-size packages, as for instance the millimetre-scaled motes provided by the SmartDust project [6].

4.10 Business Process Infrastructures

Business Process Infrastructures (BPIs) are software infrastructures supporting the specification, design and enactment of business processes. In the global economy, businesses are constantly changing their structure and processes according to market dynamics. Therefore BPIs need to be able to self-organise, namely adapt their functionality to support changes in business process requirements. The majority of BPIs are based on the agent metaphor. Agents provide flexibility and adaptability and therefore they are particularly suitable for realising self-organising infrastructures. Self-Organising BPIs are characterised by three main features: The underlying model which is used to conceptualise the business operations, the software paradigm used to develop the necessary software components and the method used to engineer the self-organising and emergent properties. Based on the underlying business model, self-organising BPIs can be classified as complex interactive BPIs, holonic BPIs, and organisational BPIs.

Complex Interactive BPIs. Complex interactive BPIs are increasingly used to support business processes, for example in dynamic workflow management [13] and intelligent manufacturing scheduling [47]. They are based on distributed software components executing without central top-down control and possibly in an asynchronous manner. Each component may be designed with different goals and structure, such that the resulting behaviour is not practically deducible from a specification of its parts in a formal way. Such systems can exhibit emergent behaviour with unpredictable results. For example, an emergent phenomenon in complex interactive BPIs supporting supply chain management is that the variation in the demand experienced by a low-tier supplier is much wider than that experienced by the OEM, sometimes increasing by as much as two times from one tier to the next [45]. Complex Interactive BPIs can be developed in terms of autonomous agents interacting in groups according to local rules. The global emergent behaviour results from local interactions between agents.

In such systems, non-linearity is considered as the main cause of emergent behaviour [33, 48]. There are various sources for non-linearity in business systems. For example in manufacturing systems three main sources of nonlinearity are capacity limits, feedback loops, and temporal delays [45].

Holonic BPIs. The Holonic BPIs aim to support *holonic enterprises* [42]. This concept has emerged from the need for flexible open, reconfigurable models, able to emulate the market dynamics in the networked economy, which necessitates that strategies and relationships evolve over time, changing with the dynamic business environment.

The main idea of the holonic enterprise model stems from the work of Arthur Koestler [35]. Koestler postulated a set of underlying principles to explain the self-organising tendencies of social and biological systems. Starting from the empirical observation that, from the Solar System to the Atom, the Universe is organised into self-replicating structures of nested hierarchies, intrinsically embedded in the functionality of natural systems, Koestler has identified structural patterns of self-replicating structures named *holarchies*. Koestler proposed the term holon to describe the elements of these systems, which is a combination of the Greek word *holos*, meaning "whole", with the suffix *on* meaning "part" as in proton or neuron [49]. Holarchies have been envisioned as models for the Universe's self-organising structure, in which holons, at several levels of resolution in the nested hierarchy [14], behave as autonomous wholes, and yet, as cooperative parts for achieving the goal of the holarchy. As such, holons can be regarded as nested agents. In such a nested hierarchy, each holon is a sub-system retaining the characteristic attributes of the whole system. What actually defines a holarchy is a purpose around which holons are clustered and subdivided in sub-holons, at several levels of resolution, according to the organisational level of dissection required.

A holonic enterprise is a holarchy of collaborative enterprises, where each enterprise is regarded as a holon and is modelled by a software agent with holonic properties, so that the software agent may be composed of other agents that behave in a similar way, but perform different functions at lower levels of resolution. A holon represents, as well, an autonomous and co-operative entity of a holonic enterprise, which includes operational features, skills and knowledge, and individual goals. Holons represent both physical and logical entities such as production departments and machines. A holon has information about itself and the environment, containing an information processing part, and often a physical processing part. An important feature of holonic enterprises is that a holon can be part of another holon, for example, a holon can be broken into several others holons, which in turn can be broken into further holons, which allows the reduction of the problem complexity.

Self-organisation of the system is achieved by using appropriate configuration meta-models of various types, such as centrally optimised or supervisory configuration and dynamic reconfiguration based on experimental data. To determine the optimum holonic structure various evolutionary and genetic approaches can be used. For example, the fuzzy-evolutionary approach, proposed by Ulieru in [49], clusters the entities of a holonic enterprise into a dynamically configured optimal holonic structure. It mimics self-organisation and evolution of natural systems as follows. On one side, self-organisation is induced by *minimising the entropy*, measuring the information spread across the system, such that equilibrium involving optimal interaction between the system's parts is reached. On the other side, it enables system's evolution into a better one by enabling interaction with external systems found via *genetic search strategies* (mimicking mating with most fit partners in natural evolution), such that the new system's optimal organisational structure (reached by minimising the entropy) is better than the one before evolution.

The holonic enterprises paradigm provides a framework for information and resource management in global virtual organisations by modelling enterprise entities as software agents linked through the Internet [55]. In this parallel universe of information, enterprises enabled with the above mechanism can evolve towards better and better structures while at the same time self-organising their resources to optimally accomplish the desired objectives. Encapsulating the dynamic evolutionary search strategy into a mediator agent and designing the virtual clustering mechanism by the fuzzy entropy minimisation strategy above, empowers the holonic enterprises with self-adapting properties. Moreover, the holonic enterprise evolves like a social organism in Cyberspace, by mating its components with new partners as they are discovered in a continuous, incremental improvement search process. Latest applications have shown high suitability for this strategy to the design of Internet-enabled soft computing holarchies for telemedicine, for example e-Health, telehealth and telematics [50].

Organisational BPIs. Organisational BPIs aim to support business in the context of global economy. The difference between Complex Interactive BPIs, Holonic BPIs and Organisational BPIs is that the latter view businesses from a global economy perspective and are focused on business theoretic models of self-organisation, such as marketing, management, and economic models.

For example, one such model from the marketing domain is the one-to-one variable pricing model [27] which refers to providing an individual offer to each customer using Internet technologies. The model involves self-organisation of the marketing policies by changing customers targeted and the prices quoted based on market dynamics, customer characteristics and the business goals. An example of a company using this type of marketing is FedEx [53]. The company allows customers to access computer systems, via the Web site, to monitor the status of their packages. For corporate customers FedEx provide software tools that enable the organisation to automate shipping and track packages using their own computing resource. Each customer is offered different prices depending on a variety of parameters. Many websites, such as eBay, also apply variable pricing for their offers.

Another example of an Organisational BPI model from the area of management is the theory of activity described in [52]. The view of a company is that it consists of networks of working groups that can change their structure, links and behaviour in response to business requirements. The aim is to capture the self-organisation decisions that need to be taken during the business operations both by managers and by interactions between employees with emphasis to solving potential conflicts of interests both of the inner and outside co-operative activity of the company.

In other cases the focus is on the technological part of the infrastructure. For example Helal and Wang in [28] propose an agent-based infrastructure and appropriate interaction protocols that would be useful in negotiating service bundles for the different services offered by agents. In that approach, agents are organised in e-business communities (Malls). Within a community, highly relevant agents group together offering special e-Services for a more effective, mutually beneficial, and more opportune e-Business. Self-organisation is based on a hierarchy of brokers which control membership and operations of the different communities. The aim of the broker hierarchy is to achieve scalability of the system and interoperability of the various heterogeneous e-business components.

5 Conclusion

There is currently a growing interest in biologically inspired systems, not only from researchers but also from industry. Recent interest by IBM, as part of their *Autonomic Computing* [34] program, and by Microsoft, as part of the *Dynamic Systems Initiative*, indicates the importance of self-organisation for managing distributed resources. In addition, future applications based on invisible intelligent technology will be made available in clothes, walls, or cars, and people can freely use it for virtual shopping, micro-payment using e-purses, or traffic guidance system [20]. These applications, by their pervasive, large-scale, and embedded nature exhibit self-organisation characteristics, and would gain in robustness, and adaptability if considered and programmed as self-organising systems.

This survey shows that most current techniques used for designing self-organising applications are direct translations of natural self-organising mechanisms, such as: the immune system, brain cells, magnetic fields, or the stigmergy paradigm. Translating natural mechanisms into software applications is a first step. However, there is a need to define specific mechanisms for self-organisation that fit electronic applications. Recent examples in that direction are given by the TOTA infrastructure (subsection 3.4), or those that can be built on the notion of tags (subsection 3.3). The challenges to take up in this field go beyond interaction mechanisms and middleware technologies favouring self-organisation. They relate to the establishment of whole software engineering methodologies encompassing design, test, and verification, based on these mechanisms as well as on sound mathematical theories enabling the definition of local goals, given the expected global behaviour.

Acknowledgements

The research described in this paper is partly supported by the EU funded Agentcities.NET project (IST-2000-28384). The opinions expressed in this paper are those of the authors and are not necessarily those of the EU Agentcities.NET partners. Giovanna Di Marzo Serugendo is supported by Swiss NSF grant 21-68026.02.

References

[1] `http://forge.gridforum.org/projects/ogsi-wg`. 9

[2] `http://www.entropia.com/`. 9

[3] `http://www.gnutella.com/`. 9

[4] `http://www.kazaa.com/`. 9

[5] `http://www.ud.com/`. 9

[6] SmartDust project, `http://robotics.eecs.berkeley.edu/~pister/SmartDust/`. 13

[7] Visual behaviors for mobile robots project, `http://www.isr.ist.utl.pt/vislab/projects.html`. 12

[8] Y. Bar-Yam. *Dynamics of Complex Systems*. Perseus Books, Cambridge, MA, 1997. 2, 3

[9] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies on the Sciences of Complexity. Oxford University Press, UK, 1999. 2, 10

[10] E. Bonabeau, V. Fourcassié, and J.-L. Deneubourg. The phase-ordering kinetics of cemetery organization in ants. Technical Report 98-01-008, Santa Fe Institute, 1998. 5

[11] S. Camazine, J.-L. Deneubourg, Nigel R. F., J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological System*. Princeton Studies in Complexity. Princeton University Press, 2001. 2

[12] N. Carriero and D. Gelernter. Tuple Analysis and Partial Evaluation Strategies in the Linda Compiler. In *Second Workshop on Languages and Compilers for Parallel Computing*, pages 115–125. MIT Press, 1989. 6

[13] Q. Chen, M. Hsu, U. Daya, and M. Griss. Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation. In C. Sierra, M. Gini, and J. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 255–263. ACM Press, 2000. 13

[14] J.H. Christensen. Holonic manufacturing systems: Initial architecture and standards directions. In *Proceedings of the First European Conference on Holonic Manufacturing Systems*, 1994. 14

[15] P. Ciancarini. Multiagent coordination: A computer science perspective. In Y. Demazeau, editor, *Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'01)*, 2001. 6

[16] P. Ciancarini, A. Omicini, and F. Zambonelli. Multiagent system engineering: The coordination viewpoint. In N. R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, volume 1757 of *LNAI*, pages 250–259. Springer-Verlag, 2000. 6

[17] J.-L. Deneubourg, S. Goss, N. Francks, A. Sendova-Francks, C. Detrain, and L. Chretien. The dynamic of collective sorting: Robot-like ants and and-like robots. In J. A. Meyer and S. W. Wilson, editors, *Proceedings of the First International Conference on Simulation and Adaptive Behavior: From Animals to Animat*, pages 356–363. MIT Press, 1991. 5

[18] G. Di Caro and M. Dorigo. Ant colonies for adaptive routing in packet-switched communications networks. In *Proceedings of PPSN V - Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *LNCS*. Springer-Verlag, 1998. 10

[19] G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, editors. *First International Workshop on Engineering Self-Organising Applications (ESOA'03)*, 2003. 1

[20] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.-C. Burgelman. Scenarios for Ambient Intelligence in 2010. Technical report, Institute for Prospective Technological Studies, 2001. 16

[21] G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35(3):66–71, 2002. 10

[22] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996. 11

[23] N. Foukia, S. Hassas, S. Fenet, and J. Hulaas. An intrusion response scheme: Tracking the source using the stigmergy paradigm. In *Proceedings of the Security of Mobile Multi-Agent Systems Workshop (SEMAS'02)*, July 2002. 11

[24] D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985. 7

[25] P. P. Grassé. La reconstruction du nid et les interactions inter-individuelles chez les bellicositermes natalenis et cubitermes sp. la théorie de la stigmergie: essai d'interprétation des termites constructeurs. *Insectes Sociaux*, 6:41–83, 1959. 1, 2, 5

[26] D. Hales and B. Edmonds. Evolving Social Rationality for MAS using "Tags". In J. S. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo, editors, *Second International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 495–503. ACM Press, 2003. 6

[27] G. Hardaker and G. Graham. *Energizing your e-Commerche through Self*. Salford, 2002. 15

[28] S. Helal, M. Wang, and A. Jagatheesan. Service-centric brokering in dynamic e-business agent communities. *Journal of Electronic Commerce Research (JECR), Special Issue in Intelligent Agents in E-Commerce*, 2(1), February 2001. 15

[29] O. Holland. Multi-agent systems: Lessons from social insects and collective robotics. In Sandip Sen, editor, *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, pages 57–62, March 1996. 6

[30] O. Holland and C. Melhuis. Stimergy, self-organization, and sorting in collective robotics. *Artificial Life*, 5(2):173–202, 1999. 2, 5, 6

[31] T. Itao, T. Suda, and T. Aoyama. Jack-in-the-net: Adaptive networking architecture for service emergence. In *Proceedings of the Asian-Pacific Conference on Communications*, 2001. 9

[32] N. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998. 8

[33] S. Kauffman. *At Home in the Universe - The Search for the Laws of Self-Organization and Complexity*. Oxford University Press, 1997. 13

[34] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003. 16

[35] A. Koestler. *The Ghost in the Machine*. Arkana, 1967. 14

[36] T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, 3rd edition edition, 2001. 4

[37] K. Lagus, T. Honkela, S. Kaski, and T. Kohonen. WEBSOM for textual data mining. *Artificial Intelligence Review*, 15(5/6):345–364, December 1999. 5

[38] S. Lynden and O. F. Rana. Coordinated learning to support resource management in computational grids. In *Proceedings of 2nd IEEE Conference on Peer-to-Peer Computing (P2P 2002)*, pages 81–89. IEEE Computer Society, 2002. 9

[39] M. Mamei and F. Zambonelli. Self-Organization in MultiAgent Systems: a Middleware approach. In G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, editors, *First Workshop on Engineering Self-Organising Applications (ESOA'03)*, 2003. 7

[40] M. Mamei, F. Zambonelli, and L. Leonardi. Co-fields: Towards a unifying approach to the engineering of swarm intelligent systems. In *3rd International Workshop on Engineering Societies in the Agents World (ESAW)*, number 2577 in LNCS, pages 68–81. Springer-Verlag, 2003. 7

[41] S. Martino. A mobile agent approach to intrusion detection. Technical report, Joint Research Center - Institute for Systems, Informatics and Safety, June 1999. 11

[42] P. McHugh, G. Merli, and W. A. Wheeler. *Beyond Business Process Reengineering: Towards the Holonic Enterprise*. John Wiley and Sons, 1995. 13

[43] F. Nedelec, T. Surrey, and E. Karsenti. Self-organisation and forces in the microtubule cytoskeleton. *Current Opinion in Cell Biology*, 15(2):118–124, 2003. 3

[44] G. O'Hare and N. R. Jennings. *Foundations of Distributed Artificial Intelligence*. John Wiley and Sons, New York, USA, 1996. 8

[45] H. V. D. Parunak and R. S. VanderBok. Managing emergent behavior in distributed control systems. In *Proceedings of ISA Tech '97, Instrument Society of America*, 1997. 13

[46] S. Russel and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice-Hall, 1995. 8

[47] W. Shen and D. H. Norrie. Agent-based systems for intelligent manufacturing: a state-of-the-art survey. *Knowledge and Information Systems, an International Journal*, 1(2):129–156, May 1999. 13

[48] M. Stewart. *The Coevolving Organization*. Decomplexity Associates LtD, 2001. 13

[49] M. Ulieru. Emergence of holonic enterprises from multi-agent systems: A fuzzy-evolutionary approach. In V. Loia, editor, *Soft Computing Agents: A New Perspective on Dynamic Information Systems*, pages 187–215. IOS Press, 2002. 14

[50] M. Ulieru. Internet-enabled soft computing holarchies for e-health applications. In L. A. Zadeh and M. Nikravesh, editors, *New Directions in Enhancing the Power of the Internet*. Springer-Verlag, 2003. To appear. 15

[51] P. Valckenaers, H. Van Brussel, M. Kollingbaum, and O. Bochmann. Multi-agent coordination and control using stigmergy applied to manufacturing control. In M. Luck, V. Marìk, O. Stepánková, and R. Trappl, editors, *Multi-Agent Systems and Applications, 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School, EASSS 2001, Selected Tutorial Papers*, volume 2086 of *LNCS*, pages 317–334. Springer-Verlag, 2001. 12

[52] V. A. Vittikh and P. O. Skobelev. Multi-agents systems for modelling of self-organization and cooperation processes. In *XIII International Conference on the Application of Artificial Intelligence in Engineering*, pages 91–96, 1998. 15

[53] K. Werbach. Syndication - the emerging model for business in the internet era. *Harvard Business Review*, pages 85–93, May-June 2000. 15

[54] I. Wokoma, L. Sacks, and I. Marshall. Biologically inspired models for sensor network design. In *London Communications Symposium*, 2002. 12

[55] H. Zhang and D. H. Norrie. Holonic control at the production and controller levels. In *Proceedings of the 2nd International Workshop on Intelligent Manufacturing Systems*, pages 215–224, 1999. 15

Self-Organizing MANET Management

Sven A. Brueckner and H. Van Dyke Parunak

Altarum Institute
3520 Green Court, Ann Arbor, MI 48105, USA
{sven.brueckner,van.parunak}@altarum.org

Abstract. In recent years, mobile ad-hoc networks (MANETs) have been deployed in various scenarios, but their scalability is severely restricted by the human operators' ability to configure and manage the network in the face of rapid change of the network structure and demand patterns. In this paper, we present a self-organizing approach to MANET management that follows general principles of engineering swarming applications.

1 Introduction

The management of mobile ad-hoc networks (MANETs) presents unique challenges that may overwhelm traditional network management approaches. Such networks are highly dynamic, severely constrained in their processing and communications resources, distributed and decentralized. Thus, centralized management approaches requiring accurate and detailed knowledge about the state of the overall system may fail, while decentralized and distributed strategies become competitive.

Designing decentralized and distributed applications in an environment as dynamic, noisy and unpredictable as MANETs requires an approach that is robust, flexible, adaptive and scalable. Self-organizing systems of agents with emergent system-level functions offer these features, but it is often far from obvious how the individual agent processes need to be designed to meet the overall design goal.

In this paper we first present a concrete management problem in the MANET domain. Then we discuss the nature of self-organizing applications in general and propose a set of design principles. In section 4 we offer a solution to the MANET management problem based on fine-grained agents dynamically interacting in the network environment. We review in section 5 how the proposed design principles are reflected in this particular self-organizing application and we conclude this paper in section 6.

2 A MANET Management Problem

Assume a network of (randomly) moving nodes that may communicate within a limited range, that may fail temporarily, and that may host client and server processes. Every node carries a client and some nodes carry a server process. A server provides a stateless and instantaneous service to a client upon request if there exists a communications path between the client and the server and if the server node

© Springer-Verlag Berlin Heidelberg 2004

is currently active. Servers in our model have no capacity constraints, and may serve as many clients at the same time as requests arrive.

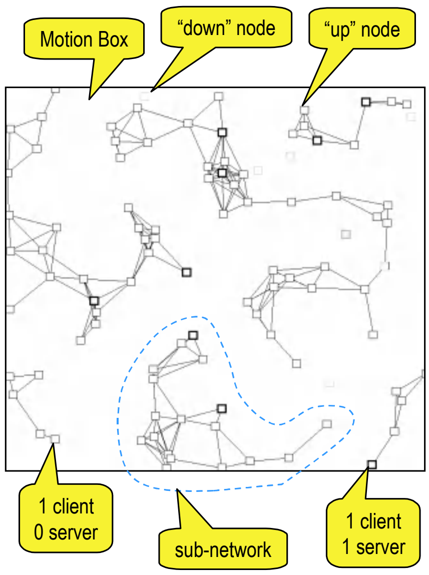


Fig. 1. Application domain: ad-hoc network of (randomly) moving nodes with power and communications limitations

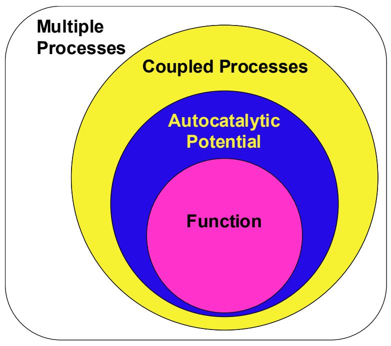


Fig. 2. Three steps to self-organization

The design goals are three-fold: for the current topology of the network determined by node locations, communications ranges and node availability, decide

- which server nodes should actually expend battery power to execute the server process;
- to which server node a particular client should send its next service request; and
- where to relocate server nodes to meet the current demand by the clients.

Thus, the network must be provided with mechanisms that self-diagnose the current network state (e.g., breaking of connections, availability of new connections, failure of nodes) and provide the information in a way that enables it to self-conFig. the ongoing processes for optimal performance.

Table 1. Categories of Information Exchange

		Topology of Inter-Agent Relationships	
		Centralized (between Distinguished and Subordinate agents)	**Decentralized** (among Peer agents)
Information Flow	**Direct** (messages between agents)	***Construction*** (Build-Time) ***Command*** (Run-time)	***Conversation***
	Indirect (non-message interaction)	***Constraint***	***Stigmergy*** (generic) ***Competition*** (limited resources)

3 Engineering Self-organizing Applications

In engineering a self-organization application such as a MANET, it is helpful to think in terms of imposing three successive restrictions on the space of all possible multi-process systems, outlined in Fig. 2.

- The various processes must be *coupled* with one another so that they can interact.
- This interaction must be self-sustaining, or *autocatalytic*.
- The self-organizing system must produce *functions* that are useful to the system's stakeholders.

In discussing each of these, we first review the concept and its mechanisms, then discuss design principles to which it leads. Elsewhere [10], we offer further discussion of these and other issues for engineering self-organizing systems.

3.1 Coupled Processes

Agents must exchange information if they are to self-organize. Different patterns of information exchange are possible, and can be classified along two dimensions: Topology and Information Flow (Table 1).

The Topology dimension depends on two different kinds of relations that agents can have with one another. When the agents can say "No" to one another within the rules of the system, they are "peer agents." When one of them (say agent A) can say "No" to the other (B), but B cannot say "No" to A, we call A the "distinguished agent" and B the "subordinate." These concepts can be developed more formally through dependency and autonomy theory [3, 8]. Centralized information exchange is between a distinguished and a subordinate agent, while decentralized information exchange is between peer agents.

The Information Flow dimension relies on environmental state variables that the agents can manipulate and sense. All information exchange is ultimately mediated by the environment, but the role of the environment is sometimes not modeled explicitly. The information flow from one agent to another is Direct if no intermediate manipulation of information is modeled, and Indirect if it is explicitly modeled.

Decentralized indirect (stigmergic) mechanisms have a number of advantages, including simplicity, scalability, robustness, and environmental integration. Four design principles support coupling among processes, and in particular stigmergic coupling.

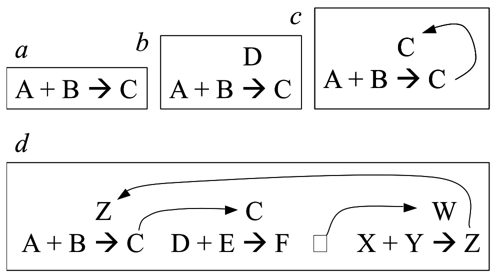


Fig. 3. Relations among Processes.--a) A simple reaction. b) D catalyzes the conversion of A and B to C. c) An autocatalytic reaction. d) An autocatalytic set of processes (shown as a ring, but other topologies are possible)

Coupling 1: Use a distributed environment. Stigmergy is most beneficial when agents can be localized in the environment with which they interact. A distributed environment enhances this localization, permitting individual agents to be simpler (because their attention span can be more local) and enhancing scalability.

Coupling 2: Use an active environment. If the environment supports its own processes, it can contribute to overall system operation. For example, evaporation of pheromones in the ants' environment is a primitive form of truth maintenance, removing obsolete information.

Coupling 3: Keep agents small. Agents should be small in comparison with the overall system, to support locality of interaction. This criterion is not sufficient to guarantee locality of interaction, but it is a necessary condition. The fewer agents there are, the more functionality each of them has to provide, and the more of the problem space it has to cover.

Coupling 4: Map agents to Entities, not Functions. Choosing to represent domain entities rather than functions as agents takes advantage of the physical fact that entities are bounded in space and thus have intrinsic locality. Functions tend to be defined globally, and making an agent responsible for a function is likely to lead to many non-local interactions.

3.2 Autocatalytic Potential

The concept of autocatalysis comes from chemistry. A catalyst is a substance that facilitates a chemical reaction without being permanently changed. In autocatalysis, a product of a reaction serves as a catalyst for that same reaction. An autocatalytic set is a set of reactions that are not individually autocatalytic, but whose products catalyze one another. The result is that the behaviors of the reactions in the set are correlated with one another. If reaction A speeds up (say, due to an increased supply of its reagents), so does any reaction catalyzed by the products of A. If A slows down, so do its autocatalytic partners. This correlation causes a decrease in the entropy of the overall set, as measured over the reaction rates. Fig 3 summarizes these concepts.

Not all coupled processes are autocatalytic. Autocatalyticity requires a continuous flow of information among the processes to keep the system moving. If the product of process A catalyzes process B, but process B's products have no effect (either directly or indirectly) on process A, the system is not autocatalytic. Furthermore, a system might be autocatalytic in some regions of its state space but not in others.

It is natural to extend this concept from chemistry to any system of interacting processes, such as a multi-agent system. A set of agents has *autocatalytic potential* if in some regions of their joint state space, their interaction causes system entropy to decrease (and thus leads to increased organization). In that region of state space, they are *autocatalytic*.

Two points are important to understand about autocatalyticity.

1. In spite of the reduction of entropy, autocatalyticity does not violate the Second Law of Thermodynamics. The rationalization is most clearly understood in the stigmergic case (Table 1). Entropy reduction occurs at the macro level (the individual agents), but the dissipation of pheromone at the micro level generates more than enough entropy to compensate. This entropy balance can actually be measured experimentally [9].
2. Information flows are necessary to support self-organization, but they are not sufficient. A set of coupled processes may have a very large space of potential operating parameters, and may achieve autocatalyticity only in a small region of this space. Nevertheless, if a system does not have closed information flows, it will not be able to maintain self-organization.

The need to achieve autocatalysis leads to three design principles.

Autocatalysis 1: Think Flows rather than Transitions. Computer scientists tend to conceive of processes in terms of discrete state transitions, but the role of autocatalysis in supporting self-organization urges us to pay attention to the flows of information among them, and to ensure that these flows include closed loops.

Autocatalysis 2: Boost and Bound. Keeping flows moving requires some mechanism for reinforcing overall system activity. Keeping flows from exploding requires some mechanism for restricting them. These mechanisms may be traditional positive and negative feedback loops, in which activity at one epoch facilitates or restrains activity at a successive one. Or they may be less adaptive mechanisms such as mechanisms for continually generating new agents and for terminating those that have run for a specified period (programmed agent death).

Autocatalysis 3: Diversify agents to keep flows going. Just as heat will not flow between two bodies of equal temperature, and water will not flow between two areas of equal elevation, information will not flow between two identical agents. They can exchange messages, but these messages carry no information that is new to the receiving agent, and so cannot change its state or its subsequent behavior. Maintaining autocatalytic flows requires diversity among the agent population. This diversity can be achieved in several ways. Each agent’s *location in the environment* may be enough to distinguish it from other agents and support flows, but if agents have the same movement rules and are launched at a single point, they will not spread out. If agents have different experiences, *learning* may enable them to diversify, but again, reuse of underlying code will often lead to stereotyped behavior. In general, we find it useful to incorporate a *stochastic element* in agent decision-making. In this way, the decisions and behaviors of agents with identical code will diversify over time, breaking the symmetry among them and enabling information flows that can sustain self-organization.

3.3 Function

Self-organization in itself is not necessarily useful. Autocatalysis might sustain undesirable oscillations or thrashing in a system, or keep it locked in some pathological behavior. We want to construct systems that not only organize themselves, but that yield structures that solve some problem. There are two broad approaches to this problem, broadly corresponding to the distinction in classical AI between the scruffy and the neat approaches. As the use of self-organizing systems matures, a hybrid of both approaches will probably be necessary.

One approach, exemplified by work in amorphous computing [1, 6], is to build up, by trial and error, a set of programming metaphors and techniques that can then be used as building blocks to assemble useful systems.

An alternative approach is to seek an algorithm that, given a high-level specification for a system, can compute the local behaviors needed to generate this global behavior. State-of-the-art algorithms of this sort are based not on *design*, but on *selection*. Selection in turn requires a system with a wide range of behavioral potential, and a way to exert pressure to select the behaviors that are actually desired.

One way to ensure a broad range of behavioral potential is to construct nonlinear systems with formally chaotic behavior. The basic idea is to let the chaotic dynamics explore the state space, and when the system reaches a desirable region, to apply a small control force to keep the system there [7]. It may seem that chaos is a complicated way to generate potential behaviors, and that it would be simpler to use a random number generator. In fact, virtually all such generators *are* in fact nonlinear systems executing in their chaotic regime.

In a multi-agent system, the key to applying this generate-and-test insight is finding a way to exert selective pressure to keep the system balanced at the desired location. Natural systems have inspired two broad classes of algorithm for this purpose: synthetic evolution, and particle swarm optimization.

Synthetic evolution is modeled on biological evolution. Many different algorithms have been developed [4], but they share the idea of a population of potential solutions that varies over time, with fitter solutions persisting and less fit ones being discarded. The variational mechanisms (which usually include random mutation) explore the system’s potential behaviors, while the death of less fit solutions and the perpetuation of more fit ones is the control pressure that selects the desired behavior.

Particle swarm optimization [5] is inspired by the flocking behavior of birds. In adaptations of this behavior to computation, solutions do not undergo birth and death (as in evolutionary mechanisms). Instead, they are distributed in some space (which may be the problem space, or an arbitrary structure), and share with their nearest neighbors the best solutions they have found so far. Each solution entity then adjusts its own behavior to take into account a blend of its own experience and that of its neighbors.

Market-based bidding mechanisms are a variation on particle swarm optimization. The similarity lies in selection via behavioral modification through the exchange of information rather than changes in the composition of the population. The approaches differ in the use that is made of the shared information. In the particle swarm, agents imitate one another, while in bidding schemes, they use this information in more complicated computations to determine their behavior.

The need to adjust the system's function to meet requirements leads to three design principles.

Function 1: Generate behavioral diversity. Structure agents to ensure that their collective behavior will explore the behavioral space as widely as possible. One formula for this objective has three parts.

1. Let each agent support multiple functions.
2. Let each function require multiple agents.
3. Break the symmetry among the agents with random or chaotic mechanisms.

The first two points ensure that system functionality emerges from agent interactions, and that any given functionality can be composed in multiple ways. The third ensures a diversity of outcomes, depending on which agents join together to provide a given function at a particular time.

Function 2: Give agents access to a fitness measure. Agents need to make local decisions that foster global goals[11]. A major challenge is finding measures that agents can evaluate on the basis of local information, but that correlate with overall system state. In one application, we have found the entropy computed over the set of behavioral options open to an agent to be a useful measure of the degree of overall system convergence [2] that agents can use to make intelligent decisions about bidding in resource allocation problems.

Function 3: Provide a mechanism for selecting among alternative behaviors. If an adequate local fitness metric can be found, it may suffice to guide the behavior of individual agents. Otherwise, agents should compare their behavior with one another, either to vary the *composition* of the overall population (as in synthetic evolution) or to enable individual agents to vary their *behavior* (as in particle swarm optimization).

4 Emergent MANET Management

A fine-grained, self-organizing agent system can solve the service location problem specified in section 2. Our agent system demonstrates the application of the design principles proposed in section 3. In section 4 we review how our design reflects these principles. Before we present our self-organizing design, we discuss a baseline solution that achieves the globally optimal service performance, but does not satisfy the severe resource constraints inherent to MANETs.

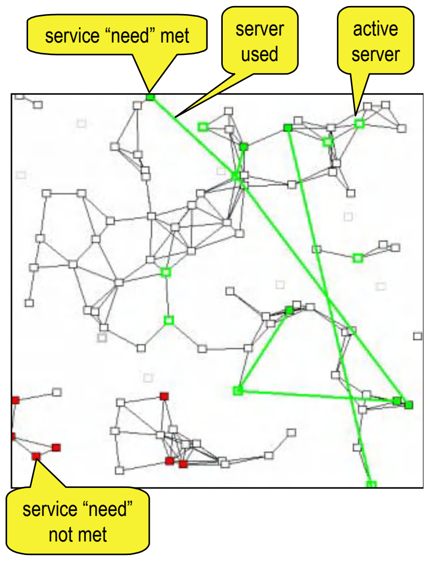


Fig. 4. A global solution provides maximum performance with unacceptable resource drain

A Global Solution

We can easily construct a global solution that provides the highest possible request-success rate for the clients. We ignore the desire to preserve battery power and let all server nodes execute the server process at all times (maximum server availability). We use global knowledge (requiring very large bandwidth) to determine for a client that wants to send a request, which available server nodes are currently in range (path exists), and then we select the recipient of the request randomly from this set.

This solution formally avoids sending requests to servers that are out of reach, whose node is currently down, or whose server process is currently not executing. But its resource requirements are too large to meet the severe constraints of the application domain (ad-hoc mobile wireless network among battery-powered nodes). Also, from a more programmatic point of view, this solution does not demonstrate emergent cognition, since the complexity of the individual node (client) is as high as the system-level complexity. Nevertheless, this solution provides us with a performance and resource-usage baseline against which we measure our local approach in the demonstration.

A Local Solution

Our local solution starts with the following initial conditions:

- Server processes shut down immediately if no requests arrive.

- A client does not know about the location of servers in the network, unless the client is co-located with a server on the same node.
- Server nodes move randomly.

Thus, in terms of our design goals, we preserve maximum battery power, but most clients' service needs are not met since they don't know which server to address.

We now define a co-evolutionary learning process based on individual reinforcement, in which the server population learns to maintain an appropriate number of active server processes and the client population learns to direct the requests to these active servers.

Server Activation Learning

Any server node is aware of the incoming requests from one or more clients. If the server process is running, then these requests are served, otherwise they fail, but the node will immediately start up the server process to be available for any new requests in the next cycle. While the server process is running, it tracks the number of incoming requests. If there are no requests, it will begin a countdown. It will either abort the countdown if new requests arrive (and are served), or shut down if it reaches the end of the countdown.

Initially, the duration of the countdown is zero. Thus, server processes are shut down as soon as no new requests come in. We define the following simple reinforcement learning process to adjust the duration of the next countdown:

(+) If a request from a client arrives and the server process is down, we increase the length of the countdown period for subsequent countdowns, since apparently the server should have been up and we lost performance (failed to serve a request) while the server was down.

(–) If no request arrives while the count down proceeds and the server process reaches the end of the countdown, then we decrease the length of the countdown period for subsequent countdowns, since apparently the server could have been down already and we wasted resources (battery power) while the server was up.

Thus, driven by the demand pattern as it is perceived at the particular server node, the server process learns to maintain the optimal availability. In effect, the server learns the mean time between requests and adjusts its countdown length accordingly to stay up long enough. With this learning mechanism in place, the client population will now assume the role of the teacher as it generates a demand signal that leads some servers to stay down (extremely short countdown) while others stay consistently up (extremely long countdowns).

Client Preference Learning

Initially, only clients that are co-located with a server on the same node have any information about possible server addresses. These clients will become the source of knowledge of the client population as they share this information with their neighbors.

Knowledge Representation. Clients manage their knowledge about and evaluation of specific servers in a dynamic set of scorecards, one for each server they know. A scorecard carries the address of the server, a score in favor (*pro*) and a score against (*con*) using this server. The current score of a server is computed as *pro - con*.

Decision Process. When a client needs to select a server, it will normalize the current scores of all scorecards so that they add up to one and select a server with a probability equal to the normalized score (roulette wheel selection). Thus, servers with a low current score compared to others have a lower probability of being chosen by the client. If the client currently does not have any scorecards, then it can only contact a server if it co-located with one, otherwise its service need will not be met in this decision cycle.

Information Sharing. If a client selects a server on a node that is currently within reach, it will send a request to the server and share the outcome of this interaction with its direct neighbors. If the request is met, the client will increase its own *pro* score of that server by one and it will send the same suggestion to its direct neighbors. If the request is not met, the *con* scores are increased in the same way. These suggestions to the neighbors may lead to the creation of new score cards at those neighbors if they had not known about this server before. Thus knowledge about relevant servers spreads through the network driven by the actual use of these servers. Furthermore, the success or failure of the interaction with a server reinforces the preferences of the client population and thus (with a random component to break symmetries) dynamically focuses the attention on a few active servers while encouraging de-activation for others (see "Server Activation Learning").

Truth Maintenance. The constant change of the network topology, driven by the node movements and their failures, requires that the client population continuously update its knowledge about reachable servers and their evaluation. While the score-sharing mechanism ensures that the performance of a reachable server is continuously re-evaluated, the clients still need a mechanism to forget references to servers that do not exist anymore or that are out of reach now. Otherwise, in long-term operation of the system, the clients would drown in obsolete addresses.

We implemented a mechanism that reverses the traditional approach to truth maintenance. Rather than maintaining any knowledge until it is proven wrong, we begin to remove knowledge as soon as it is no longer reinforced. This approach is successfully demonstrated in natural agent systems, such as ant colonies, where information stored in pheromones begins to evaporate as soon as it is laid down. A client "evaporates" its scores (*pro* and *con* individually) by multiplying them with a globally fixed factor between zero and one in each decision cycle. Thus, both scores approach zero over time if the client or its neighbors do not use the server anymore. If both scores have fallen below a fixed threshold, then the scorecard is removed from the client's memory – the client forgets about this server.

A client also chooses to forget about a particular server, if the *con* score dominates the *pro* score by a globally fixed ratio ($con / (con + pro) >$ threshold > 0.5). Thus, servers that are trained by the client population to be down are eventually removed from the collective memory and are left untouched. They only return into the memory of clients if all other servers have also been forgotten and their co-located client is forced to use them.

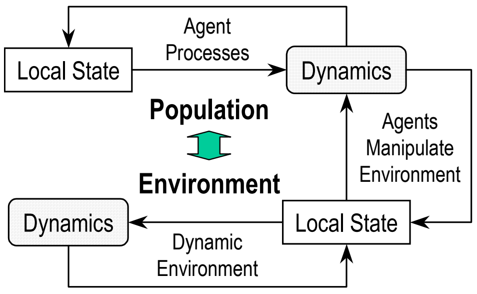


Fig. 5. Stigmergic coordination of agents in a shared environment

Server Node Location Learning

In a co-evolutionary process, the server and client populations learn which clients should focus on which servers. We can stabilize this preference pattern and reduce the need for re-learning by decreasing the likelihood that the connection between a client and its chosen server is disrupted. Since the risk for a disruption of the path between a client and a server generally increases with the distance between their nodes, moving the server node towards its current clients will decrease this risk.

We assume that any client and server processes have means to estimate their respective node's current spatial location and that the server node may actually control its movement within certain constraints if it chooses to.

As a client sends a request to a server, it includes its current location in the request message. The server node computes the vector between the client and the server location and adds up all vectors from all requests within a decision cycle. Vectors of requests that failed are negated before they are added to the sum. The resulting combined vector determines the direction of the next move of the server node. If the requests failed because the server process was down, then the node moves away from the "center of gravity" of the clients that contacted this server. Otherwise, the node will move toward these clients. The length of the step for the server node is fixed to a global constant, characterizing the physical ability of the node to move.

Stigmergic Coordination

The coordinated behavior of many simple agents (server, client, node) in the highly dynamic and disruptive environment of the Service Location demonstration emerges from peer-to-peer interactions in a shared environment driven by simple rules and dynamic local knowledge. The individual components of the system are not explicitly aware of the overall system functions of self-diagnosis and self-reconfiguration.

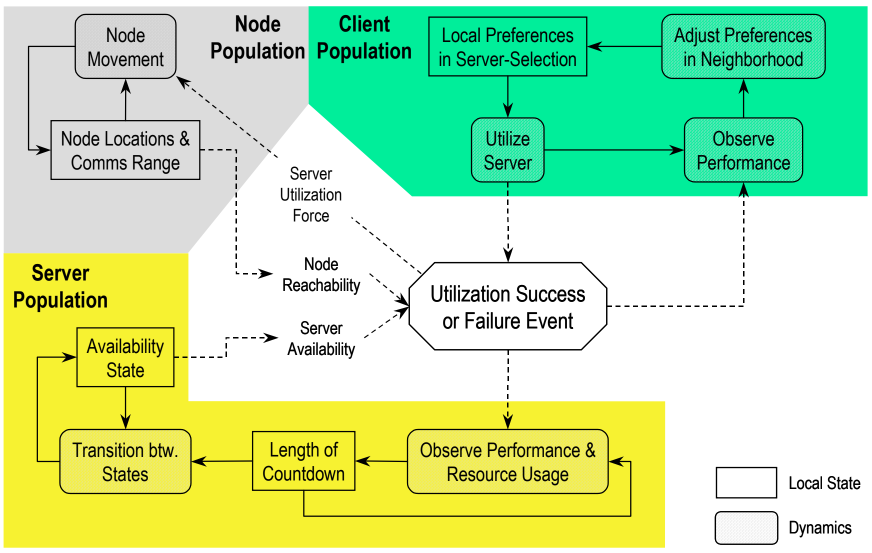


Fig. 6. Stigmergic coordination of MANET agent populations

The coordination mechanism detailed in this demonstration is an example of stigmergy, in which individual agent activity is influenced by the state of the agent and its local environment. As agent activity manipulates the environment, subsequent agent activity dynamics may change (Fig. 5). If this flow of information between the agents through the environment establishes a feedback loop that decreases the entropy of the options of the individual agents, then coordinated behavior emerges in the population. We engineer the agent behavior and the indirect information flow, so that the emergent coordinated behavior meets the design goal.

Three populations of processes (agents) contribute to the emerging system functionality. Because each population operates in the shared network environment, the other populations influence its dynamics. For instance, the clients coordinate their server choice through the exchange of scores, but their ability to focus on only a few servers depends on the server population's ability to identify the emerging intention of the clients and to maintain the server processes on the correct nodes. Fig. 6 identifies the main flow of information among the three populations driven by their respective dynamics and linked by the occurrence of successful or failed utilization events requests from clients to servers.

Comparison with the Global Solution

We implemented both solutions (global and local) in a software demonstration that allows us to observe the ongoing operation of the system visually, but which does not provide any systematic experimentation (parameter sweep, formal analysis) opportunities.

Let us consider the static case first. We reduce the probability of nodes to fail to zero and freeze their random motion. Thus, the MANET topology remains fixed to the initial state. With the global solution, clients that are in reach of a server immediately find the server and succeed in utilizing the service, while all others continually fail. The performance of the system in providing the service to the client is maximal. Since the servers are always on, the usage of processing power by the server processes is maximal too.

Our local solution begins with a very low performance, since all clients except those that are co-located with a server on the same node do not have any knowledge about available servers. But once a co-located client initiates its first service request to its local server, the knowledge about this server rapidly propagates through the server's sub-network and other clients start utilizing the server too. If there are more than one server in a sub-network, then the clients' asymmetric reinforcement of their scorecards and the servers' activation learning quickly breaks symmetries and focuses the clients' request onto one server. At this point, the performance of the system is as good as with the global solution, but the resource usage is lower if the network topology provides for multiple servers in a sub-network.

Let us now introduce a change in the topology as either a node fails or random movement changes the link structure. Such a change may or may not change the reachability or availability of servers for a sub-population of clients. The more clients are affected, the more severe are the effects of this topology change. With the global solution, the performance of the system immediately adjusts to the maximally possible utilization success rate. In contrast, our local approach experiences a temporary drop-off in performance as the affected clients first try to utilize servers that are no longer available and then exchange knowledge about other locally available servers that had shut down. The more severe the topology change is, the larger is the aggregated loss of performance during this re-learning cycle.

Finally, let us repeatedly trigger topology changes. These changes occur stochastically and we hypothesize that the frequency of changes of a fixed severity follows a power law, where more severe changes are exponentially less frequent. As the duration of the re-learning cycle increases with the severity of the changes, there will be a point where the next change occurs before the re-learning is completed. We expect a clear phase change in the performance (and thus applicability) of our local solution as the dynamics of topology change surpass a threshold. We hope to verify this prediction in future research.

5 Reviewing the Approach

In this section we briefly review how our design of the service location system meets the design principles proposed in section 3.

Coupling 1: Use a distributed environment. The interactions among clients are constrained by the current topology of the network. A client may exchange information only with its direct neighbors. Through this restriction we ensure that any change in the network topology immediately changes the clients' interaction pattern without requiring reasoning about the topology.

Coupling 2: Use an active environment. The clients and servers maintain their individual virtual environment in which they maintain dynamics analogous to pheromones. A client's set of scorecards of known servers and their performance is its environment: it deposits scores, which are evaporated over time, and it selects the next server based on the currently "observed" score pattern. A server's virtual environment is the variable that represents the duration of the next countdown.

Coupling 3: Keep agents small. Clients and servers are very simple in their knowledge representation, reasoning and interaction processes. The complexity of the management system emerges from the continuous interactions of the agents and the dynamics of the environment.

Coupling 4: Map agents to entities, not functions. We identify client and server agents with existing nodes interacting across local communication links rather than with service allocation functions.

Autocatalysis 1: Think flows rather than transitions. Knowledge about active servers flows from client to client driven by their individual attempts at utilizing a particular server.

Autocatalysis 2: Boost and Bound. The length of the next countdown of a server is boosted by a failed utilization attempt and bounded by countdowns that run to the end. Clients' attempts at utilizing a server may boost the population of clients that may access this server (flow of server location knowledge), but this population size is bounded to the size of the sub-network (topology) and the observed performance of the server.

Autocatalysis 3: Diversify agents to keep flows going. Clients will forget about a server if they do not utilize it for a while or if its performance is low. Information about these forgotten servers will flow again, if the client that is co-located with the server is forced to utilize it when it runs out of all other options.

Function 1: Generate behavioral diversity. Clients and servers have different behaviors, and the system requires both populations to achieve the management function. Without clients' learning to prefer a server subset, no server could ever deactivate, and without servers' learning to shut down, clients could not focus on a small server subset.

Function 2: Give agents access to a fitness measure. The global goal of the management function is to shut down as many servers as possible while maintaining a sufficient service rate. Servers perceive the current fitness of the system against this goal by the failure of utilization requests while they are shut down (performance too low) or by the completion of a countdown without another incoming request (resources wasted). Clients seek to maximize system performance by focusing on only a few active servers.

Function 3: Provide a mechanism for selecting among alternative behaviors. Clients share their knowledge about servers with their neighbors, thus providing them with alternative behaviors (direction of utilization attempt). Through the individual scorecard learning, only those behaviors that are actually successful will sustain their presence in the population . Clients balance the exploration of new behaviors with the exploitation of successful behaviors through their probabilistic selection of servers.

6 Conclusion

Using self-organization and emergence to engineer system-level functionality may be advantageous in many application domains, but often it is not obvious how to design the underlying processes to achieve the desired function. In this paper we introduced a set of general design principles that reduce the overwhelming set of options at design decision points and thus guide the engineer towards a successful application design.

We demonstrate the design approach in the real-world example of a MANET management system. MANETs meet many criteria of application domains in which self-organization and emergence of functions is required to provide the necessary robustness and scalability. These networks are highly **D**ynamic, processing and communication is **D**ecentralized, and local decision makers are **D**istributed and **D**eprived of resources.

Our self-organizing agent system for MANET management performs close to the optimum achieved by a global coordination mechanism that could not realistically be implemented on a real-world network since it does not meet the resource constraints. The agents in our system rely on stigmergy to coordinate their activities.

Acknowledgements

This work is supported in part by the DARPA Knowledge-Plane seedling study, contract N00014-03-M-0252 to Altarum, under DARPA PM Christopher Ramming. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

References

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous Computing. Communications of the ACM, 43(5):74-82, 2000. citeseer.nj.nec.com/abelson95amorphous.html.

[2] S. Brueckner and H. V. D. Parunak. Information-Driven Phase Changes in Multi-Agent Coordination. In Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003), Melbourne, Australia, pages 950-951, 2003. http://www.erim.org/~vparunak/AAMAS03InfoPhaseChange.pdf.

[3] C. Castelfranchi. Founding Agent□s□ □Autonomy□on Dependence Theory. In Proceedings of 14th European Conference on Artificial Intelligence, Berlin, Germany, pages 353-357, IOS Press, 2000.

[4] C. Jacob. Illustrating Evolutionary Computation With Mathematica. San Francisco, Morgan Kaufmann, 2001.

[5] J. Kennedy, R. C. Eberhart, and Y. Shi. Swarm Intelligence. San Francisco, Morgan Kaufmann, 2001.

[6] MIT. Amorphous Computing Home Page. 2003. Web Site, http://www.swiss.ai.mit.edu/projects/amorphous/.

[7] E. Ott, C. Grebogi, and J. A. Yorke. Controlling Chaos. Physical Review Letters, 64(11):1196-1199, 1990.

[8] H. V. D. Parunak. Distributed AI and Manufacturing Control: Some Issues and Insights. In Y. Demazeau and J.-P. Mller, Editors , Decentralized AI, pages 81-104. North-Holland, 1990.

[9] H. V. D. Parunak and S. Brueckner. Entropy and Self-Organization in Multi-Agent Systems. In Proceedings of The Fifth International Conference on Autonomous Agents (Agents 2001), Montreal, Canada, pages 124-130, ACM, 2001. www.erim.org/~vparunak/agents01ent.pdf.

[10] H. V. D. Parunak and S. A. Brueckner. Engineering Swarming Systems. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, Editors, Methodologies and Software Engineering for Agent Systems, pages (forthcoming). Kluwer, 2003. http://www.erim.org/~vparunak/MSEAS03.pdf.

[11] D. Wolpert and K. Tumer. Collective Intelligence. 2002. Web site, http://ic.arc.nasa.gov/projects/COIN/index.html.

Toward the Application of Self Organizing Systems to Agent Infrastructures – A Critical Review and Prospects

Simon Thompson

BT Exact Technology, Intelligent Systems Lab
BT Adastral Park, pp12, MLB1, Ipswich, IP5 3RE
simon.2.thompson@bt.com

Abstract. This paper examines the case for using self organizing systems in elements of infrastructures for the operation of Agents, Autonomous Agents and Service Orientated Systems. An overview of the use of self organizing systems in different elements of the infrastructures of an agent based system is provided. There is also a discussion of the ways in which these techniques can be brought into the mainstream of agent implementation and deployment.

1 Introduction

Agent Based Systems (ABS) are the subject of extensive research and development by many groups and under many different guises (Service Orientated Architectures, Web Services, Autonomous Agents, Intelligent Agents, Semantic Grids, The Semantic Web). The FIPA standards body [20] has proposed a set of standards and infrastructures to support the deployment, integration and operation of ABS infrastructures and this has been implemented by many different teams to support various applications. The best known FIPA implementation is JADE [5]; however, there are at least 20 open and closed source implementations that have now been developed.

Figure 1 shows a schematic of what the FIPA specification suggests as a structure for a ABS. Messages can be routed between agents via a messaging channel (1), the addresses of agents and service providers can be discovered in a name service and a directory facilitator (2) and messages can be sent between agents directly if their current address has been obtained (3). The Agentcities project [53] has attempted to use these toolkits and standards in open trials recently. In addition a large and growing number of non-standard, proprietary toolkits for developing and deploying ABS are available; and toolkits that support other standards such as OGSA [21] and Web Services.

However, at the time of writing there are very few reported successes of fielded ABS or even of open web service deployments. In addition, the fielded exemplars of ABS are in controlled corporate environments, in contrast to the open environments that are often envisioned. Why is this?

A possible reason is the cost of managing ABS infrastructures. The process of deploying most software involves the initialization of one component on one

© Springer-Verlag Berlin Heidelberg 2004

machine. Business cases for the upgrade or installation of new computer systems are often built on the simplification of the system and the reduction of management costs. The installation and deployment of the current generation of ABS infrastructures is complex and their management is time consuming.

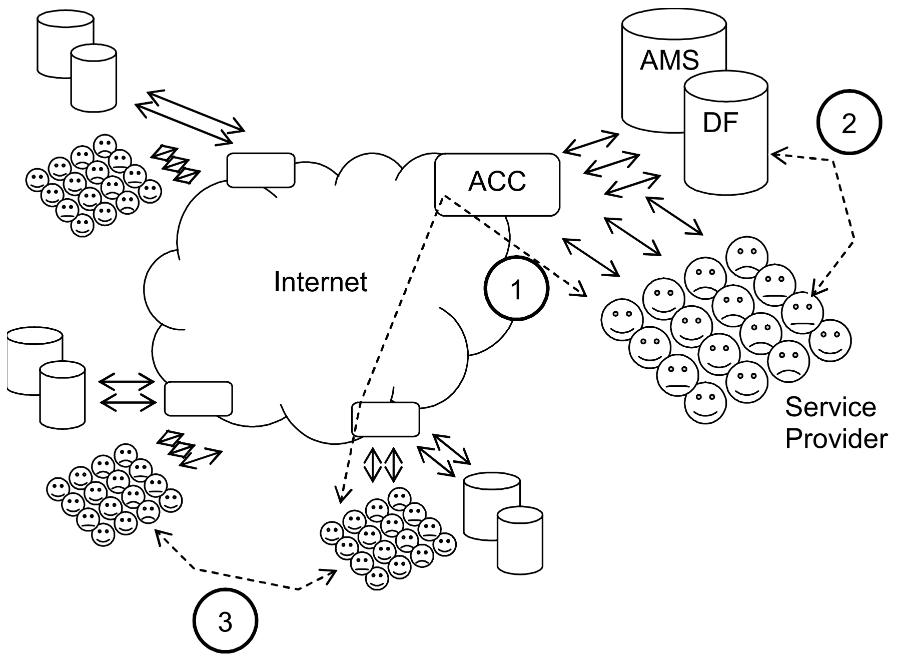


Fig. 1. Schematic of a FIPA infrastructure for an ABS

In the long term, ABS will fulfill their potential as a technology only when they are deployed widely and in open networks, and when the structures, scale and operational properties (online, responsive, heavily used) of ABS reflect our wider economy, technology and society. Visions of this sort of deployment large scale can be found in the Agentlink Roadmap [34] produced for the European Union. We assert that this challenge probably cannot be met without using principles, techniques and algorithms derived from the study of self organizing systems.

The expectation of the Agentlink project, and others [7][24][17] is of millions of agents deployed in tens of thousands of applications running on millions of devices in geographically distributed networks, interacting in real time to obtain knowledge, actions and resources from one another. The interdependency and interaction characteristic of such a system seems likely to lead to an explosion of complexity. Managing the comparatively simple, predictable and modular internet based systems that are currently deployed stretches the budgets of even the largest organizations, and the technical skills of the most sophisticated users, so without new solutions to these problems there is little prospect of ABS fulfilling their potential.

1.1 Self Organization and Agent Based System Infrastructures

A widely accepted definition of self-organization [33] is that structure emerges without explicit pressure or involvement from outside the system and that the constraints on the form of structure of the system are internal to its implementation. Self-organizing systems may have attractors; states or structure into which they tend to settle over time. A problem is solved, or the self-organizing system is harnessed to our use by mapping the problem components to the initial states of the system and then allowing the primitive interactions that drive the self organization to take their course. The solution to the problem or the desired state of the target system is then read from the state of the self organizing components [48]. Figure 2 illustrates this idea, showing a notional system in which forces representing the quality of service provided by a data store act on notional objects that represent agents attempting to establish which data store they should use.

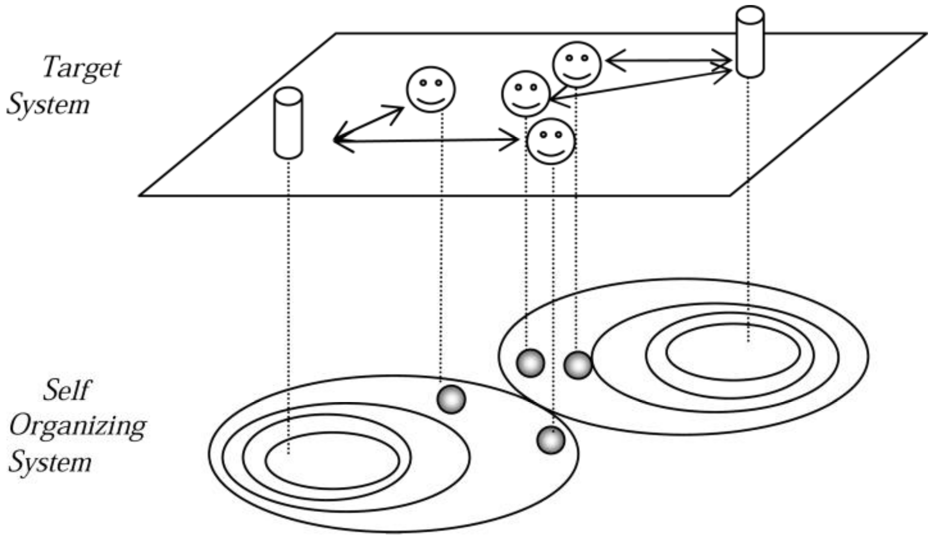


Fig. 2. Map from self-organizing system to target agent infrastructure component

Categorization of self-organizing system types is problematic, because researchers have explored a vast range of metaphors and principles to develop systems with self organizing behaviour. While an exhaustive classification is not attempted here it is possible to identify three basic metaphors or mechanisms that are used by many of the self organizing systems that have been reported:

- Rule or game theory based systems: autonomous agents make decisions about the actions that they will take according either to strictly responsive rules of behavior or to calculations of the effect of their behavior on their own operation or on the systems future behavior. The result of these independent actions, behaviors and calculations is some overall global behavior. For example, a group of agents could make a decision to send resources to a particular agent, independently, and without a request by that agent, because they become aware that the deficiency of resource for that agent endangers it and places the stability of the whole system at risk.

Sophisticated agents would calculate the amount of resource that they will send to the needy agent based on the belief that other agents in the system will also realize that such action is required and will act in a similar way. These agents are organized by mutual observation, modeling and inference; they do not talk or signal to each other, but they observe, model and act on each others behavior to produce a system. Systems of this sort are often modeled in terms of Game Theory, with Nash Equilibrium in which an initial action by one agent gets a reaction by all other agents that in turn elicits the initial action; the best response of all actors elicits the best response of all other actors [41] being used to establish the attractor to which the system is to gravitate. Alternatively the system may draw inspiration from swarm or ant style systems [10].

- Artificial physics based systems operate a set of global rules that produce an overall system behavior. Agents operate within a system of forces that limit and channel their actions; their local behavior is dependent on their resolution of these forces. The global behavior of the system is dependent on the way that the forces operate. Often in these systems an agent is modeled as exerting a force on other agents. In this way an agent may find that it is difficult to push its way to the front of the most popular messaging port in its system. Using another, less popular channel is easier, but by doing so it repels other agents from attempting the same thing. Systems of this type can be thought of as attempting to organize disparate agents in the way that a magnet organizes iron filings. Analogies used to drive such systems include electric charge, immune response, magnetism, sound, photophilia, flocking and swarming, and spin glass systems [26]
- Competitive systems operate from a different metaphor. In this case the agents compete for some notional resources with the side effect that the desired system behavior is achieved. We differentiate this from evolutionary or selective systems where external selection according to some externally derived criteria, not inherent in the system is applied to move a system toward one of several attractors. For example, in this case a system may treat access to the processor as a drain on the utilizing agent and it may treat successful activity on the processor (solutions to some problem discovered, information clicked on) as energy or food for the agent. Agents that fritter processor time away are modeled as not being energetic enough to force their way to the front of the queue again, and so are shut out of the system by agents that use it successfully. Eventually weak agents run out of energy and are killed by the system. Competitive systems can be thought of as organizing agents in the way that a forest organizes plants. Analogies that drive such systems are market economics [28], macro-economic resource competition[27] and models of eco-systems [29].

The distinctions between these system types are arguable, but describing them here conveys the kind of approaches that can form part of the tool kit for the implementation of an agent infrastructure.

Alternatively we can classify systems based on their role and status in the infrastructure; they can be used peripherally to make specific decisions; they can be part of the operation or they can be the fabric of the system.

The self-organizing system can be used to make decisions as to which of a choice of infrastructure component (directory, database, message channel) to use; the self organizing system is not being used as a direct solution, but to manage access to the components that solve the problem. For example, we can use some technique (rules, competition, physics) to decide who will act as the naming database in the system at a particular time. In this way a failover mechanism that can operate in the face of a crash or network problem is provided, and the user does not have to make an explicit decision about which data store to choose.

The self-organizing system can be used as part of the operation of the components providing infrastructure. For example, a network of servers acts as a point of contact for directory information, caching answers locally and propagating new information across the network. Servers can choose to answer queries based on criteria that are tuned to reflect the properties that the infrastructures designer has decided are important. For instance the system may be constructed to be able to be substantially degraded (lose a lot of nodes) before it fails to be able to provide a high quality of service. Alternatively the system may guarantee that if a particular request for information can be satisfied from the information that is registered in it then it will always respond correctly to that request; if the question is not answered then it is because it is not answerable.

Finally the most extreme position is to use the self-organizing system directly to provide the infrastructure. Each agent in our directory system is nominated to be a directory agent and to accumulate information at that agent in order that it is as independent as possible from the rest of the system. Of course, this is maximally inefficient, but it is also maximally robust, and it is also simple to maintain, deploy and manage agents of this type. The system does not have to be booted in a specific order, or user agents do not have to wait for directory components to become available before registering themselves.

1.2 Organization and Scope

In this paper we will address only the question of how self-organizing systems can be used to construct and support ABS infrastructures. We are not concerned with how self organizing systems can be used for application development, and this topic is not addressed. Furthermore, there is no discussion of how value can be provided to users in the terms of their direct use of the system. For example, we do not consider how using a self organizing system will improve the range of functions available to a patient when he or she accesses health care services. Instead this paper will look at how the non-functional properties of a health care system implemented using ABS can be enhanced or provided using self organizing algorithms and techniques.

Specifically we will examine the following opportunities for the application of self-organizing systems.

- Directory services and structures: how address resolution can be supported by a self organizing system to improve responsiveness, eliminate centralization bottlenecks and single points of failure and improve scalability.
- Message routing and delivery: how the best routes between agents deployed on various devices and networks can be discovered, repaired and managed.
- Load balancing and platform management: how access to the computing surfaces that support agent societies can be managed by using self organizing systems.

- Other possible applications including authentication and security: how reputation and authority can be established by an environment without the use of a centralized system and how semantic definitions can be maintained across ABS using the principles of self organizing systems.

This paper is organized as follows. The next five sections discuss the various parts of an agent infrastructure and the problems that they present that can be addressed using self-organizing systems. Section 2 will discuss the use of self-organization for developing directory services. Section 3 will discuss how message transport systems can be enhanced, Section 4 describes how processor utilization and load balancing can be supported by self-organizing techniques. Section 5 describes how self-organizing systems can be applied to other aspects of ABS infrastructure.

The paper is concluded with a discussion of the directions that the Agent Based Systems Infrastructure community should take now in order to develop these techniques and realize their value for users.

2 Directory Services

Using self-organizing systems to provide the directory services in an ABS infrastructure is an obvious use for these techniques. Providing reliable, searchable, secure and scalable directory services is not something that has traditionally been well addressed in agent infrastructures, and self-organizing mechanisms seem to offer an ideal solution.

FIPA specifies that a compliant architecture should have two directories; the agent Naming Service (ANS) and the Directory Facilitator (DF). The ANS provides an information store for discovering the address of named service providers, where as the DF provides the addresses of service providers that provide particular services. Most proposed ABS infrastructures have similar features; a white pages directory to do name search and a yellow pages directory to perform service lookup. These information services are provided by middle agents that manage the query and edit interfaces to them via appropriate published API's. Additional middle agents are sometimes provided to fulfill functions that are related to the expected requirements of the particular systems that will be generated by the ABS to use the infrastructure.

These schemes are centralized. The agents in a system must register their services with one or more central directories. Various schemes for overcoming this are proposed. The FIPA-DF uses federated searches to answer queries [20][5]; the Zeus toolkit [15] provides a tree based naming infrastructure and RETSINA [30]has been implemented with a distributed search algorithm to discover names. However, in all these cases particular agents hold the addressing and reference information that is necessary to the functioning of all the other agents in the system, this information is not propagated or replicated and each time it is required the agent that holds it is contacted again. This introduces bottlenecks in processing, especially for tasks that require large scale delegation and negotiation, that limit the scalability of the system and it introduces points of failure when directory agents crash or fail to reply to messages in time.

How can a self organizing system provide some remedy to these issues?

- The structure of the network that holds the information can be organized to remove bottlenecks and points of failure which can isolate particular agents.
- The information in the network can be propagated to reduce the average time that it takes to find it.
- The information in the network can be propagated to ensure that particular queries will always be correctly answered.

A typical, some what naïve solution was implemented to extend the Zeus Agent Toolkit. The algorithm is shown in detail in Figure 4. A schematic that illustrates its function and purpose is shown in Figure 3. To summarize each platform in the system maintains a list of *n* other agent platforms that it can contact to ask for either the address of another agent, or for a list of agents that can provide a particular service. When a platform receives a request for a name or for a service that it cannot competently answer (if it hasn't received updates in a time out period) then it consults its list of *n* contact agents for the information and stores the replies as an update before forwarding them to the querying agent. Additionally each platform maintains a list of other agents, a contact list that it will provide to other platforms when asked about joining the network. This list is maintained to be as disjoint as possible from the list that is used to query the network. In this way when a platform joins the network it is given a list of platforms that it will query which is as distant as possible from the platforms that the originating platform will be querying. When a contact agent is queried in this way it then attempts to update its list of contacts from the network by asking for most recently registered agents to provide as contacts (and excluding any that it acted as contact agent for). In this way the network topology is pushed to be bushy and distributed as new agents tend to be directed to query other new agents when they join the network.

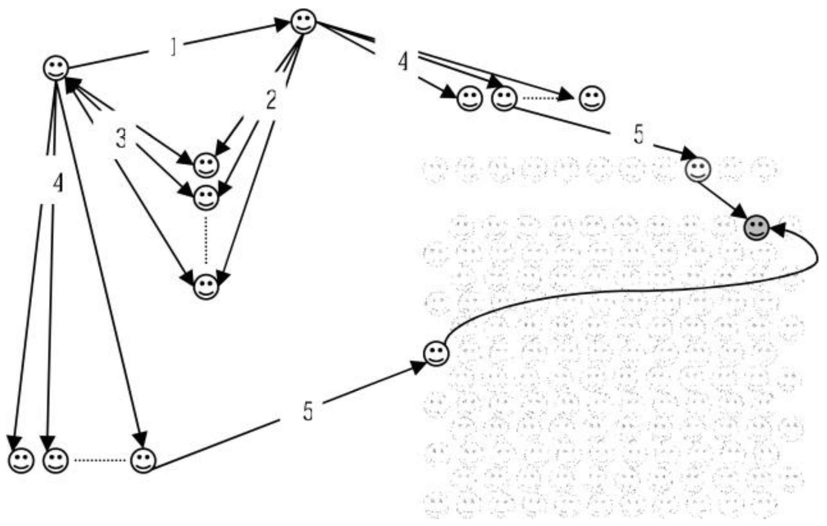


Fig. 3. Schematic of a simple self organizing directory query system. An agent joining the system sends a query to any other agent (1) the reply contains the addresses of a set of contacts (2) to use when building a query forwarding list (3) when queries are made (4) different discovery routes are likely (5).

When a query is made it is propagated through the network and the size of the propagation made is limited with a TTL token (time to live) which is decremented by each propagating node. When a TTL reaches 0 the propagation stops. This mechanism was implemented by the GNUTELLA system, and it is prone to many problems; nodes can cheat the system by setting TTL too high, improperly implemented nodes can fail to decrement the token, these issues can cause system flooding and loss of service. In our mechanism nodes are expected to probalistically decrement the TTL on the basis of the load that the forwarding node is placing on the network, limiting access to the directory service resource.

The algorithm implemented in Zeus is an early example of how a simple heuristic (providing new agent registrations to joining agents from well known points of contact) can result in desirable system properties (a bushy network with few super-nodes to act as bottle necks; fair access to the network constrained by the individual nodes preventing flooding by not forwarding queries). However, considerably more sophisticated examples and implementations have been developed by a number of teams.

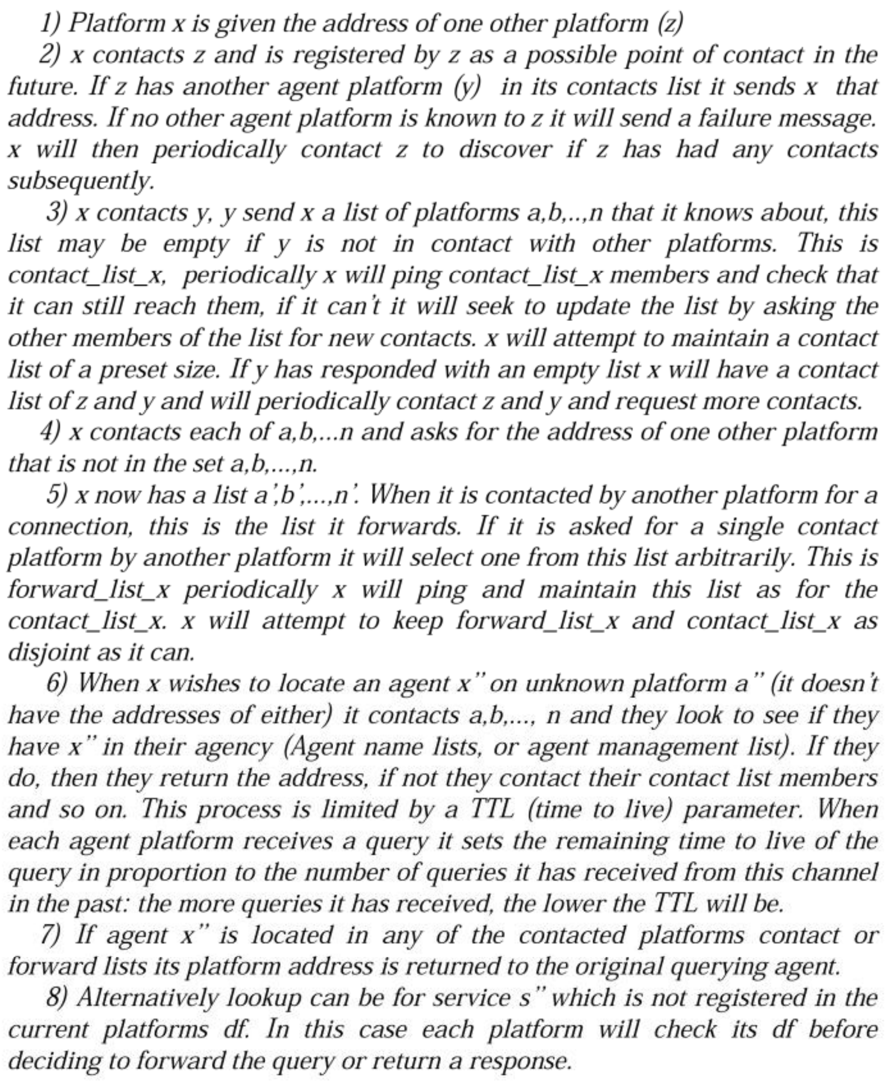
The DIET system [35] uses a different approach. DIET agents us a hashing mechanism for naming that allows agents to communicate with selected subsets of the agents on the network. A DIET agent has a name that consists of two components, a tag name and a family name. The tag name is unique, the family name is a hash code generated by the application launching the agent into the platform. The platform then addresses agents according to these identifiers, so, for example, the part of the hash code that identifies auction agents can be used as a destination address by an application agent and the platform will send that message to all auction agents on the network. This is a nice example of ëdge□intelligence. The infrastructure is not responsible for determining which agents belong to which groups □it will forward messages to all the agents that it determines the message addresses. The application developer is responsible for selecting a family name that will result in only appropriate messages being forwarded. Pastry [44] uses a mechanism similar to that employed by DIET to discover where a resource is on the network via an indexing space.

NEVERLATE [12] is an index based system that maintains a topologically organized map of nodes and propagates directory information from source nodes to other nodes in order to ensure that it is never more than a certain number of hops from a querying node.

CAN [43](Content Addressable Network) provides a discovery system based on a mesh of nodes and uses a hash-based lookup of a pointer to a resource on the network that can then be used to route a request to that resource over the network.

Clearly, self-organizing infrastructures are attractive for ABS naming and service discovery, but a number of questions remain to be answered. Using these techniques improves robustness, scalability and can eliminate bottle necks. But while we know that using some form of self organizing directory infrastructure will serve us well it is not clear what properties we should choose as the most significant for the ABS of the future.

In the case of a distributed database the property of always returning a record if it exists in the network seems essential. But is this the case in an ABS? Does it really matter if only 98% of the restaurants in a city are returned to my personal agent? We believe that millions of agents must co-habit the system and that response times of 10ths of seconds are required to construct viable applications, but will it in fact become apparent that we need networks of tens of billions of agents and response times of 100ths of seconds are needed?



1) Platform x is given the address of one other platform (z)

2) x contacts z and is registered by z as a possible point of contact in the future. If z has another agent platform (y) in its contacts list it sends x that address. If no other agent platform is known to z it will send a failure message. x will then periodically contact z to discover if z has had any contacts subsequently.

3) x contacts y, y send x a list of platforms a,b,..,n that it knows about, this list may be empty if y is not in contact with other platforms. This is contact_list_x, periodically x will ping contact_list_x members and check that it can still reach them, if it can't it will seek to update the list by asking the other members of the list for new contacts. x will attempt to maintain a contact list of a preset size. If y has responded with an empty list x will have a contact list of z and y and will periodically contact z and y and request more contacts.

4) x contacts each of a,b,...n and asks for the address of one other platform that is not in the set a,b,...,n.

5) x now has a list a',b',...,n'. When it is contacted by another platform for a connection, this is the list it forwards. If it is asked for a single contact platform by another platform it will select one from this list arbitrarily. This is forward_list_x periodically x will ping and maintain this list as for the contact_list_x. x will attempt to keep forward_list_x and contact_list_x as disjoint as it can.

6) When x wishes to locate an agent x" on unknown platform a" (it doesn't have the addresses of either) it contacts a,b,..., n and they look to see if they have x" in their agency (Agent name lists, or agent management list). If they do, then they return the address, if not they contact their contact list members and so on. This process is limited by a TTL (time to live) parameter. When each agent platform receives a query it sets the remaining time to live of the query in proportion to the number of queries it has received from this channel in the past: the more queries it has received, the lower the TTL will be.

7) If agent x" is located in any of the contacted platforms contact or forward lists its platform address is returned to the original querying agent.

8) Alternatively lookup can be for service s" which is not registered in the current platforms df. In this case each platform will check its df before deciding to forward the query or return a response.

Fig. 4. A typical algorithm for building a self organizing directory service system

3 Message Transport, Routing and Delivery

Another use for self-organizing systems in providing an infrastructure for ABS is to support the messaging infrastructure that it uses.

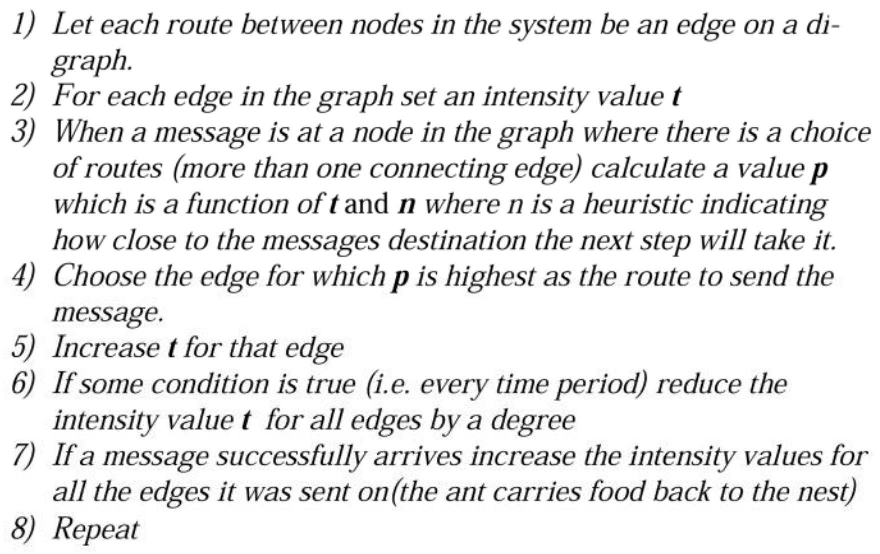
Current system implementers are faced with the need deliver messages through networks of various devices including the internet, 3GPP, 812.11, Bluetooth, intranets and other various networking protocols. These delivery routes are decentralized and non-hierarchical and ensuring reliable delivery in these environments is difficult using standard techniques.

With the introduction of ant colony based optimization systems [18] it has become possible to approach this decentralized problem with a self-organizing system [1][2][9][46][51]. A generalized example mechanism employed by these algorithms is shown in Figure 5 below.

The essential feature of these systems is that through local decision making they establish a feedback mechanism which stimulates the use of particular channels for messages that are successfully dispatched and reinforced by repeated use, and that over time new channels can take over if messages fail to get through.

Examples of other algorithms using other principles include Spiderlink [45] which uses the abstraction of dragline spinning to develop a self-organizing message routing mechanism. Spiderlink does not store information about the network in a static table, instead it uses mobile agents to transport messages and information about routing is stored in those agents. In a similar vein Minar et al. [38] show that for certain types of networks over which messages must be routed an ecology of mobile agents living inside a network can provide a solution that is superior to the tradition message routing systems previously employed. They make the point that the use of an inherently decentralized mechanism for routing messages round an inherently decentralized network has a low □impedan ce mismatch;□there is a synchronicity between the problem and the solution. Another effort is described in Awerbuch et-al [4]which uses an ant based system to route messages across networks that are being attacked in various ways.

A number of self-organizing infrastructures for message routing and distribution have been developed by the peer-2-peer research community. Probably the best known of these is Tapestry [54]. Tapestry has been developed to support distributed databases, and provides adaptive, self organizing message routing facilities and location (DOLR). Tapestry is adaptive, to dynamic changes in the network that it is searching for objects and is designed to have minimal message latency and high message throughput. Tapestry virtualizes object locations by providing a nodeID□s randomly. Objects are published and unpublished on the network and messages can then be routed to them; messages can also be routed to objects specifically present on a particular node.



1) *Let each route between nodes in the system be an edge on a di-graph.*
2) *For each edge in the graph set an intensity value **t***
3) *When a message is at a node in the graph where there is a choice of routes (more than one connecting edge) calculate a value **p** which is a function of **t** and **n** where n is a heuristic indicating how close to the messages destination the next step will take it.*
4) *Choose the edge for which **p** is highest as the route to send the message.*
5) *Increase **t** for that edge*
6) *If some condition is true (i.e. every time period) reduce the intensity value **t** for all edges by a degree*
7) *If a message successfully arrives increase the intensity values for all the edges it was sent on(the ant carries food back to the nest)*
8) *Repeat*

Fig. 5. An ant based message routing algorithm

4 Processor Utilization and Quality of Service

Managing the load on nodes in a decentralized ABS is problem which can seriously impact on system performance. For example, the release of tickets to a popular concert may result in a denial of service as hundreds of thousands of personal agents attempt to buy one. Managing the allocation of resources to tasks can be done with powerful scheduling algorithms, but these themselves consume resources and tend to be centralized in nature.

An alternative approach, founded on the principles of self-organizing systems is taken by the DIET system [36]. In DIET trigger behaviors that adapt to the system load are used to initiate processing. These triggers are more likely to fire when activity is low, but as system load increases they fire less often. Each of the kernels in the DIET system is deployed with limits on the number of threads in use and the buffer sizes available. DIET agents migrate between platforms and send messages using mechanism that are dependent on these limits, and so as load increase on a platform the QOS of the service that it offers is decreased, making it less attractive to agents that are utilizing it. The kernel guarantees service to the agent community overall, but encourages the agents within the community to adapt their behavior to prevent messages being lost, or agent transfers failing.

The Hive system [37] utilizes a similar mechanism with resource constrained cells which provide access to shadows representing resources that the agents are to utilize. The agents migrate from machine to machine in order to utilize resources and programming. Hive has a number of centralized features including the listing of the cells in the network [39].

Scheduling jobs to clusters of workstations can be organized using market based methods according to the demand levels placed on the individual processors [13]. An auction based algorithm is used to improve performance by up to 14 times for highly parallel tasks.

5 Other Applications of Self-Organizing Systems in Agent Based Infrastructures

The interface between computer systems and humans is one of the most important factors in influencing their success. In ABS where the needs of humans, their requests, preferences and orders are represented by a proxy, the agent, this is doubly true. This makes the identification and verification of the users of an ABS a critical issue for the implementers of an ABS infrastructure. In fact the self organizing aspects ABS are touted as a solution for authentication and security for distributed systems in general [23][25][14], so perhaps it makes sense to use self-organization as the mechanism for providing identity resolution and authentication in ABS infrastructures.

Agent societies are sometimes defined by the subscription of the agents to a shared ontology; this enables them to reason over a shared set of terms. However, managing the revisions and propagation of knowledge about the shifting semantic of an ontology is an issue in large societies. In [32]a system of resource and user access based on the significance by use and dependency in the network is reported for an ontology server.

6 Conclusion

It seems that ABS Infrastructures can benefit from the use of self-organizing techniques in providing naming and directory services; supporting message transport; managing information stores; organizing access to resources and providing security and authentication. Why is it then that these techniques have not been adopted in the leading ABS infrastructures and standards?

Of course, these techniques are new; they are the product of various unrelated and uncoordinated efforts. It may be that gradually we will see these techniques adopted into the mainstream toolkits and standards, in time, and by a process of trial and error and natural selection.

Acting in this way may incur a penalty; once infrastructures are adopted, widespread procurement has been made by industry and users and developers have been trained; there is no going back. If the community waits for winners to emerge the window of opportunity for these techniques may shut. If an example is needed of the possible consequences of allowing the market to decide these methods just use a computer for a few days and wait for blue screen and terse error report to appear.

To ensure that fielded ABS infrastructures do not cripple and limit the applications and benefits of agent technology our community must systematically pursue a scientific program that will :

- Establish empirically which system properties are the most significant in respect to ABS infrastructures.
- Analytically which mechanisms are the most effective at delivering the appropriate properties?
- Demonstrate the value of self-organizing systems in real world case studies business cases for the adoption of the techniques can be built.

What are the first steps in such a program?

Firstly researchers must establish the benchmarks that they are attempting to work to. This is not a trivial matter and is fraught with problems. The machine learning community, evolutionary computing community and the scheduling community have both established benchmarks which they use to determine which techniques are the most useful. In the machine learning community the UCI repository [8]is widely used to validate supervised learning systems. The evolutionary computing community uses a wide range of problems such as the royal road problem [40]and the scheduling community uses job shop scheduling and other benchmark constraint satisfaction problems[19] [49]. Recently the trading challenges developed for agent based ecommerce [50]and in robot soccer [3] have had a similar effect.

In all cases this has brought substantial benefits, but problems as well. Researchers seek to develop small incremental improvements to algorithms. Algorithms that address problems which are outside the scope of recognized test sets and problems are disregarded by researchers who need to generate publications. However, overall the development of benchmark sets has provided reproducibility, focus and rigor to these communities. Beyond the use of benchmarks it is necessary to have bounds and measures that algorithms can be analyzed mathematically against. This is crucial because it establishes a finite program: when an algorithm is provably optimal no further development is required.

In addition to agreeing benchmarks it is important that the standards of measurement that are required in studies of algorithm performance are also agreed. An example of two papers that evaluate scaling in ABS infrastructures illustrates the need for this. In [31] Lee et-al. study the Zeus infrastructure's scaling properties and conclude that the infrastructure can scale to accommodate (low) hundreds of agents. In [16] Vitaglione et-al study the JADE system and show scaling to thousands of agents. The difference in the measurements is in the experimental set-up. The JADE study examines the use of pairs or couples of agents performing round trip messaging. The Zeus study investigates contract net messaging between communities of hundreds of agents. The load in Lee's study in terms of messages exchanged is therefore higher and the two studies cannot be compared. Both studies are equally valid, and both are useful to the community, but their value would be multiplied if they were directly comparable. Again, we can learn from other communities. Machine learning has established a standard of statistical rigor for experiments [42]; the ABS infrastructure community should do the same.

In addition to research it is necessary to deliver technology to the users. Recent years have demonstrated two routes that are accessible to all for doing this. Firstly standardization provides both credibility and publicity to commercialization efforts. It is easier for industrial researchers to 'sell' technology that is the result of a serious standardization process: mostly because something that is a formal standard has at least that much substance. It is not clear that a single unified standard is the way forward.

In reality ABS infrastructure developers will have to confront a multiplicity of technologies and legacy systems in any environment. Specific components should be standardized and several different standards should be available to suit particular requirements. If there is widespread knowledge of and agreement about the particular issues that are being standardized the general principles of the components developed using the different standards should be similar enough to achieve the aim of reducing integration costs considerably.

In any case FIPA provides various standards for MTP (SC00075, SC00084); envelope representations (SC00085, SC00088) and content language (SC00008, XC0009, XC00010, XC00011). The difficulty in integrating a web service and a FIPA agent at a message level is not the result of the standards bodies concerned being different, but the fact that at a messaging level a FIPA agent can send complex expressions filled with constraints, logical expressions and implications which are hard to encapsulate in a XML document [11]

The other delivery route is via open source software. There are now dozens of open source agent infrastructures, however, the infrastructures are generally not “plug and play” and the research community tends not to produce components that can be plugged in. There are of course honorable exceptions. ATOMIK [52]is a parser and ontology management layer which is architected to be pluggable. JESS [22] is a reasoning system which can be used easily with many open source infrastructures. These kinds of components should be developed to use particular technological developments in self-organizing systems applied to ABS infrastructures, and they should be aimed to be as accessible and reusable as possible.

Taken together these two approaches will maximize the opportunities for the application and use of self-organizing agent infrastructure components. On the one hand various standards can be adapted to include these concepts; on the other hand various proprietary application developments can utilize stand alone infrastructure components. Importantly developers will not face an “all or nothing” challenge when attempting to make use of this new technology.

The final necessary component of our research program is real world case studies. More difficulties loom here. As all real world practioners know, commercial organizations are reluctant to trial and roll out new technologies in operational settings. In addition commercial considerations such as intellectual property law and commercial confidentiality prohibit dissemination of results in some cases. None the less the publication of the results of real world case studies of technology application such as [6]and those described in the survey in [47] are invaluable generating support for the application of ABS, and the development of case studies of ABS infrastructure will be necessary to persuade software suppliers to use self-organizing systems in their next generation of enterprise infrastructures.

It is clear to us that the infrastructures of future agent based systems will be founded to a greater or lesser extent on self organizing systems. The only question that remains is which of the techniques that are available to the community will prove to be most successful in the real world?

References

[1] Appleby S., Steward, S. Mobile software agents for control in telecommunications networks,□ *BT Technology Journal*, 12 (2) April 1994 pp.104-113.

[2] Arabshahi, P., Gray, A., Kassabalidis, I., Das, A. Adaptive Routing in Wireless Communication Networks using Swarm Intelligence,□ *9th AIAA Communication Satellite Systems Conf.* 17-20 April 2001, Toulouse, France.

[3] Asada, M., Veloso,M.M., Tambe, M., Noda,T., Kitano, H., Kraetzschmar, G.K. Overview of RoboCup-98,□ *AI Magazine*, Volume 21 (1). pp 9-19, 2000.

[4] Awrbush, B., Holmer,D., Rubens, H. Provably Secure Competitive Routing against Proactive Byzantine Adversaries via Reinforcement Learning.□ *Technical Report, John Hopkins University Computer Science Department.* May, 2003.

[5] Bellifemine,F., Poggi,A., Rimassa.G., JADE □A FIPA-compliant agent framework.□ *Proceedings of PAAM'99*, London, April 1999, pagg.97-108.

[6] Berger, M., Bouzid, M., Buckland, M., Lee, H., Lhuiller, N., Olpp, D., Picault, J., Shepherdson, J. An Approach to Agent-Based Service Compostion and its Application to Mobile Business Processes□ *IEEE Transactions on Mobile Computing*, July-September 2003 Vol 2(3), pp 197-206. 2003.

[7] Bernars-Lee, T., Hendler,J., Lassila, O. The Semantic Web- A new form of web content that is meaningful to computers will unleash a revolution of new possibilities,□ *Scientific American*, May 2001.

[8] Blake, C.L. & Merz, C.J. UCI Repository of mach ine learning databases,□ [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

[9] Bonabeau, E., Henaux, F., Guerin,S., Snyers, D., Kuntz,P., and Theraulaz,G., Routing in telecommunication networks with "Smart" ant-like agents. *In Proceedings of IATA'98, Second Int. Workshop on Intelligent Agents for Telecommunication Applications.* Lectures Notes in AI vol. 1437, Springer Verlag, 1998.

[10] Bonabeau, E., Dorigo,M., Theraulaz,G. Inspi ration for optimization from social insect behaviour,□ *Nature*, Vol 406, pp 32-42, 2000.

[11] Bothelo, L., Willmott,S., Zhang, T., Dale,J., A review of Content Languages Suitable for Agent-Agent Communication,□ *EPFL I&C Technical Report #200233*.

[12] Chander, A., Dawson, S., Lincon, P., Stringer-Calver, D., NEVRLATE: Scalable Resource Discovery,□ *In Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02).* pp 382-389. Berlin, Germany. May 2002.

[13] Chun, B.N. Culler, D.E., Use r-Centric Performance Analysis of Market-Based Cluster Batch Schedulers.□ In *: Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02).* pp 30-39. Berlin, Germany. May 2002.

[14] Crosbie, M., Spafford,E. Defending a co mputer system using autonomous agents,□ *18th National Information Systems Security Conference*, Oct 1995.

[15] Collis, J.C. Ndumu, D.T., Nwana, H.S., Lee, L.C. The ZEUS agent building toolkit□ *BT Technology Journal*, 16 (3) pp 60-68, July 1998.

[16] Cortese, E., Quarta, F., & Vitaglione, G. Scalability and Performance of the JADE Message Transport System,□ *AAMAS Workshop on AgentCities*, Bologna, 16th July, 2002.

[17] de Roure, D., Jennings, N.R. and Shadbolt,N. The Semantic Grid: A future e-Science infrastructure.□ *Int. J. of Concurrency and Computation: Practice and Experience* 15 (11), 2003.

[18] Dorigo M., Maniezzo, V. & Colorni, A. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26, No. 1, pp 1-13.

[19] Drummond, M. Scheduling Benchmarks and Related Resources, *Newsletter of the AAAI SIGMAN* Vol. 6 No 3. 1993.

[20] FIPA http://www.fipa.org.

[21] Foster, I., Kesselman, C., Nick, J.M., Tuecke, S. (2002) The Physiology of the Grid. An Open Grid Services Architecture for Distributed Systems Integration, The Globus Alliance. June 2002.

[22] Friedman-Hill, E., *JESS in Action: Java Rule-based Systems*. Manning Publications Co. July 2003, ISBN 1930110898.

[23] Ghanea-Hercock, R. Authentication with P2P agents *BT Technology Journal,* 21 (4), pp 148 152, October 2003. Kulwer Academic Publishers, Dodrecht, Netherlands.

[24] IBM Research. Autonomic Computing: IBM s Perspective on the State of Information Technology. AGENDA 2001 conference in Scottsdale, AZ. USA. 2001

[25] Kagal, L., Finin, T., Peng, Y. A Framework for Di stributed Trust Management, *Proceedings of the IJCAI-01 Workshop on Autonomy, delegation and control*.

[26] Kauffman., S.A. *Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

[27] Kearney,P., *The Dynamics of the World Economy Game*, Technical Report SLE7IT/93-30, December 1993.

[28] Kearney, P., Merlat, W. Modeling market-based decentralized management systems. *BT Technology Journal* 17 (4): 145-156, 1999.

[29] Langton, G.G. *Artificial Life.* Addison-Wesley. Redwood City, 1989.

[30] Langley, B., Paolucci, M., and Sycara, K., Discovery of Infrastructure in Multi-Agent Systems. *In Agents 2001 Workshop on Infrastructure for Agents, MAS, and Scalable MAS.* Available from http://www-2.cs.cmu.edu/~softagents/papers/infrastructureDiscovery.pdf.

[31] Lee,L.C., Nawana, H.S., Ndumu, D.T., De Wilde, P. The stability, scalability and performance of multi-agent systems, *BT Technology Journal*, 16 (3) July 1998.

[32] Li, Y., Thompson, S.G., Tan, Z., Giles, N., Gharib, H., Beyond ontology construction; Ontology Services as online knowledge sharing communities *In Proceedings of 2nd International Semantic Web Conference*. Lecture Notes in Computer Science 2870/2003, Springer Verlag, pp 469 483, 2003.

[33] Lucas, C. Self-Organizing Systems FAQ for Usenet newsgroup comp.theory.self-org-sys. http://www.calresco.org /sos/sosfaq.htm#1.2, 2003.

[34] Luck, M., McBurney, P., Preist, C. Agent Technology: Enabling Next Generation Computing Version 1.0, ISBN 0854 327886 (http://www.agentlink.org/roadmap/index.html)

[35] Marrow, P., Bonsma, E.,Wang, F., Hoile, C., DIET a scaleable robust and adaptable multi-agent platform for information management *BT Technology Journal*, 21 (4), pp 130 -137, October 2003. Kulwer Academic Publishers, Dodrecht, Netherlands.

[36] Marrow, P., Koubarakis,M., van Lengen, R.H., Valverde-Albacete,F., Bonsma,E., Cid-Suerio,J., Figueiras-Vidal,A.R., Gallardo-Antolin,A., Hoile,C., Koutris,T., Molina-Bulla,H., Navia-Vazquez, A., Raftopoulou,P., Skarmeas, N., Tryfonopoulos, C., Wang, F., Xiruhaki,C., Agents in Decentralised Information Ecosystems: The DIET Approach. *Symposium on Information Agents for E-Commerce, AISB'01 Convention,* 21st - 24th March 2001 University of York, United Kingdom.

[37] Minar, N., *Designing an Ecology of Distributed Agents*, MIT Masters Thesis, August 1998.

[38] Minar, N., Kramer, K.H. Maes, P. Cooperating Mobile Agents for Dynamic Network Routing. *Software Agents for Future Communications Systems*, Springer-Verlag, 1999, ISBN 3-540-65578-6.

[39] Minar,N., Gray, M., Roup,O., Krikorian, R., Maes,P. Hive: Distributed Agents for Networking Things, *In Proceedings for ASA/MA'99, the First International Symposium on Agent Systems and Appplications and Third International Symposium on Mobile Agents*. 1999.

[40] Mitchell, M., Forrest, S., and Holland, J. H. (1992). The royal road for genetic algorithms: Fitness landscapes and GA performance. In *Proceedings of the First European Conference on Artificial Life.* Cambridge, MA: MIT Press/Bradford Books.

[41] Nash, J. The Bargaining Problem, *Econometrica*, 18(2) April 1950.

[42] Provost, F.J., Fawcett, T., & Kohavi, R. Building the case against accuracy estimation for comparing induction algorithms, In: *ICML-98, Proceedings of the Fifteenth International Conference on Machine Learning*. 1998

[43] Ratnasamy, S., Francis, P., Handly, M., Karp,R., Shenker,S., A Scalable Content-Addressable network, *SIGCOM01,* San Diego, CA. USA., August, 2001.

[44] Rowston, A., Druschel, P. Pastry: Scalable distributed object location and routing for large-scale peer-2-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[45] Saffre, F., Ghanea-Hercock, R. Spiderlink: survical of the dumbest in ad-hoc networks. In: *Proceedings of Workshop on Ad hoc Communications*, Bonn, Germany, September 16th 2001.

[46] Schoonderwoerd, R., Holland, O., Bruten, J. Rothkrantz, L.J.M., Ant-Based Load Balencing in Telecommunications Networks. *Adaptive Behaviour* 5 (2), pp169-207, MIT Press, 1996.

[47] Shen, W. and Norrie, D.H. Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. *Knowledge and Information Systems*, an International Journal, 1(2), 129-156, 1999.

[48] Steels, L. Language as a complex adaptive system, Lecture Notes in Computer Science. Parallel *Problem Solving from Nature – PPSN-VI*, Schoenauer & et-al (eds) Springer-Verlag, Berlin.

[49] Taillard. E. Benchmarks for basic scheduling problems. *European Journal of Operations Research*, 64:278--285, 1993.

[50] Wellman, M.P., Greenwald, A.R., Stone, P., Wurman, P.R. The 2001 Trading Agent Competition The 2001 Trading Agent Competition. AAAI/IAAI 2002: 935

[51] White, T. Pagurek, B. Towards Multi-Swarm Problem Solving in Networks *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)* 1998

[52] Willmott, S., Constantinescu, I., Calisti, M. Multilingual Agents: Ontologies, Languages and Abstractions, In: *Proceedings of Workshop on Ontologies in Agent Systems at Autonomous Agents 2001*, Montreal, Canada 2001.

[53] Willmott, S., Somacher, M., Constantinescu, I., Dale, J., Poslad,S., and Bonnefoy, D. : The Agentcities Network Architecture, *Proceedings of the First International Workshop on Challenges in Open Agent Systems* (July 2002)

[54] Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz. J.D. Tapestry: A Resilient Global-scale Overlay for Service Deployment, To appear in *IEEE Journal on Selected Areas in Communications*. 2003

A Catalog of Biologically-Inspired Primitives for Engineering Self-Organization

Radhika Nagpal

Department of Systems Biology, Harvard Medical School
240 Longwood Avenue, Boston MA 02115, USA
rad@eecs.harvard.edu

Abstract. The Amorphous Computing project is aimed at developing programming methodologies for systems composed of vast numbers of locally-interacting, identically-programmed agents. This paper presents some of the building blocks for robust collective behavior that have emerged as part of this effort, and describes how organizing principles from multi-cellular organisms may apply to multi-agent systems.

1 Introduction

During embryogenesis, cells with identical DNA cooperate to form complex structures such as ourselves, starting from a mostly homogeneous egg. The process involves a complex cascade of spatial and functional decisions taken by individual cells. Even so, the process of development is incredibly robust to variations in individual cell behavior, varying cell size and division rates, and cell death. Many organisms develop correctly independently of initial embryo size; for example, frog embryos with half as many cells develop into half-size larva. Even after development, many retain the ability to regenerate entire structures that have been damaged. For example, newts can regenerate amputated limbs and some species of starfish can regenerate an entire new body from a limb. The ability of multi-cellular organisms to achieve such complexity and reliability is awe-inspiring for an engineer.

Emerging technologies such as MEMS devices (micro-electronic mechanical devices), are making it possible to bulk-manufacture millions of tiny computing and sensing elements and embed these into materials, structures and the environment. Applications being envisioned include smart environments where sensors are ``painted" onto walls or surfaces, reconfigurable robots/structures composed of millions of identical modules that self-assemble into different shapes to achieve different tasks, armies of ant-like robots that can collectively achieve complex global tasks [1, 2, 3, 4]. The charm of such systems is the potential of replacing specialized engineering with vast numbers of cheaply bulk-manufactured generic parts, that can achieve many purposes through programming.

This raises a fundamental question: how do we program such systems? We expect that there will be too many elements to individually program or name, and it will not be possible to control precise layout and precision interconnects. Individual agents are likely to have limited resources, limited reliability, and only local information and

© Springer-Verlag Berlin Heidelberg 2004

communication. Ideally we would like the system to produce robust global behavior in spite of individual limitations and failures, and be self-maintaining without human intervention. How do we achieve robust global behavior from vast numbers of unreliable agents? And can we abstractly describe global goals and automatically derive robust distributed agent programs?

The objective of the Amorphous Computing project is to invent programming methodologies for such multi-agent systems [5]. The earliest emphasis was on pattern-formation and self-assembly, and much of the inspiration came from multi-cellular organisms. Several simulated systems were designed that could translate user-specified patterns and shapes into robust agent programs. The agent programs did not rely on regular placement, synchronous behavior, or perfectly reliable agents [6, 7, 8].

A common methodology emerged from these examples: goals were described at a high level using programming languages based on generative rules, and these generative rules were then mapped to local programs for agent behavior [9]. In addition, a common set of simple and powerful primitives for agent interactions emerged that were sufficient to create large classes of complex structures. Two important aspects of these primitives are (1) they are generally insensitive to variations in individual behavior, numbers of agents, timing, and placement (2) it is possible to theoretically analyze their behavior. Many of these primitives were directly inspired by studies from developmental biology, i.e. how cells organize to make coordinated spatial decisions robustly during embryogenesis. As we continue to think about designing self-maintaining and self-repairing systems, the analogy to multi-cellular behavior continues to be extremely fruitful [10].

While the global-to-local programming methodology has been presented elsewhere [9], this paper is a first attempt towards assembling a catalog of primitives for multi-agent control, inspired by metaphors from multi-cellular systems.

2 Primitives for Robust Local Behavior

Biologists have long been fascinated with the ability of cells in an embryo to locally coordinate to develop into complex organisms and the ability of this process to regulate in the face of failures and natural variation. There is a large literature of conceptual ideas of how cells might achieve this robustness and complexity [11, 12] and how embryogenesis can be thought of as a program of interacting cells [13]. These concepts can provide a basis for designing robust distributed algorithms that achieve similar goals in our artificial systems.

This section discusses a set of basic building blocks that can be achieved by simple agent programs. Complexity is achieved by the composition of these building blocks in principled ways. The primitives are:

1. Morphogen gradients and positional information
2. Chemotaxis and directional information
3. Local inhibition and local competition
4. Lateral inhibition and spacing
5. Local monitoring

6. Quorum sensing and counting
7. Checkpoints and consensus
8. Random exploration and selective stabilization

The first five have been extensively used in Amorphous Computing. The remaining three are basic behaviors that may prove useful as we design systems with different types of goals. These primitives represent a small subset of the mechanisms actually used in embryogenesis; for example, they do not touch upon mechanical methods for interaction such as cell movement, differential adhesion, or local forces.

2.1 Morphogen Gradients and Positional Information

Regional specification is an important aspect of embryogenesis. Undifferentiated cells must determine what part of the structure to form. In many organisms, the embryo starts with a small population of cells as organizing centers or poles, which create gradients of diffusable factors called morphogens. Other cells in the embryo can use the morphogen concentration to determine what region they lie in relative to the pole. For example in the fruit fly embryo, poles are established at the anterior and posterior ends, and gradients of factors such as *bicoid* determine the initial segmentation into head, thorax and abdomen regions. Finer-grain segmentation occurs as these coarse regions produce other morphogens [12]. Conceptually morphogen gradients have played an important role in understanding how cells acquire positional information [14, 13].

Morphogen gradients have been a key primitive in amorphous computing systems, and similar primitives have independently emerged in sensor networks and reconfigurable robots for self-organizing position information [2, 4]. A simple agent program can produce a "morphogen gradient": an agent creates a morphogen gradient by sending a message to its local neighborhood with the morphogen name and a value of zero. Agents who hear the message forward the message to their local neighbors with the value incremented by one, and so on until the morphogen has propagated over the entire system. Each agent stores and forwards only the minimum value it has heard for a particular morphogen name. Thus the value of the morphogen increases as one moves away from the source agent, and the value reflects the distance to the source. The source may even be multiple agents. An important variation on this primitive is *active morphogens*, where the value of the morphogen decays over time. Here the source agent must constantly produce morphogen messages to maintain the gradient and the gradient adapts as the network topology changes. The morphogen primitive is easy to analyze, provides reasonable distance estimates, and is robust to many different types of variations and failures [6].

Morphogen gradients can be used to infer positional and geometric information in many ways. For example, agents can compare their local value to a threshold to determine if they are within a region relative to the source, or agents can compare different morphogens to determine where they lie relative to two sources. It is an extremely versatile primitive, and appears as part of the agent program for many of the primitives described below. In that way it can be considered to be more fundamental than the rest.

2.2 Chemotaxis and Directional Information

Chemotaxis refers to the ability of an organism to follow the gradient of a diffusing substance. There are a plethora of examples in biology where gradients are used to sense direction: neurons detect and follow chemical gradients, yeast cells create extensions in the direction of mating signals, bacteria can migrate towards higher concentrations of nutrients [11].

An agent program can be created that emulates this behavior by combining the morphogen primitive from above with the ability of agents to query their neighbors. An agent can collect the values of a morphogen in its neighborhood and by comparing these values determine the direction of the gradient. An agent can then select a neighbor agent that is closer to the source. This process can be used to create a route from an agent to the source, and an interesting aspect of this primitive is that it guarantees connectivity in the event of large regional failures. Thus unlike the previous primitive that was useful for geometric inferences, this primitive is good for direction and connectivity. This primitive was a key piece in the work by Coore on self-organizing topological patterns, and since has been extensively exploited in amorphous computing systems [7].

2.3 Local Inhibition and Local Competition

Within regions of cells, some cells may differentiate to take on special roles. For example in epithelial (skin) cells, only some cells differentiate to produce hair follicles [12]. How these cells are chosen can be thought of as a process of local competition and local inhibition. Through some random process, a cell chooses to differentiate and it simultaneously creates a local signal that inhibits neighboring cells from differentiating. The competition may also dependent on the fitness of the individual cells. A particularly interesting example of such competition is in the developing wing of the fruit fly, where fast dividing cells cause slowly dividing cells to commit apoptosis, even though the slower cells would be capable of producing the entire wing in the absence of the fast cells. The faster cells are able to signal neighboring slower cells, causing them to lose the competition [15, 16].

Local inhibition and competition are useful primitives for leader election within a spatially local group of candidate agents. A simple agent program can be designed to achieve this, by combining a random process with limited range morphogen gradients. All competing agents pick random numbers and count down. If an agent reaches zero without being interrupted then it becomes a leader and produces a morphogen that inhibits other agents from competing further. This algorithm is also simple to analyze and has been shown to be robust to asynchronous behavior and agent death and addition [5]. A variation on this primitive allows agents to create morphogens with a value related to their fitness, and agents are only inhibited by morphogens with higher fitness [8]. This primitive also has the ability to automatically adapt if the leader agent dies, since the inhibiting influence of the morphogen also disappears and the competition can resume. In Amorphous Computing this primitive has been used extensively for choosing an agent from a set of agents.

2.4 Lateral Inhibition and Spacing

Many patterns in biology exhibit beautiful regular spacing; for example, bristle and hair patterns, scales, and fruit fly eyes. Many different theoretical models have been developed to explain how such spacing is generated [12]. One such model is called lateral inhibition and is believe to be the mechanism by which bristles emerge at regular intervals. It is in fact almost identical to the previous model, except that cells produce an inhibition morphogen with a much smaller range than the extent of competition. Thus a differentiated cell may inhibit spatially local cells from differentiating, but globally the competition continues until all cells are either inhibited or differentiated.

The agent program is the same as before. Agents pick random numbers and if an agent counts down to zero without being inhibited, it creates a morphogen of range *d*. The remaining uninhibited agents continue to compete. At the end of the competition, no two leader agents will be within distance *d* of each other with high probability, and all agents will be within distance *d* of some leader. Thus a regular spacing of leader agents is produced within a field of agents. This algorithm has been used to design several simple schemes for self-organizing hierarchical structure [5].

2.5 Local Monitoring

Most organisms are capable of wound repair, which depends on the ability of neighboring cells to detect regional death and respond appropriately. Detection may be through mechanical means, such as contact, and repair may then consist of growth or recruiting migrating cells to fill in the void.

In agents, we can emulate detection of regional death through local monitoring. Each agent periodically broadcasts a message of *aliveness* to its neighbors, and keeps track of when it last heard an aliveness message from its neighbors. In the event of a missing neighbor, the agent can trigger some response. This primitive presents an interesting tradeoff. The frequency of aliveness messages from neighbors determines the speed with which faults can be detected and the lag in response time. Faster response time comes with a higher cost of communication. Local monitoring can be used to make many of the previously presented primitives capable of self-repair, such as the chemotaxis primitive. As mentioned before, the chemotaxis primitive has been used to self-organize topological patterns. By using active gradients and local monitoring, this primitive creates topological patterns that seamlessly adjust connectivity as regions of agents die or are replaced [10].

2.6 Quorum Sensing and Counting

Many single cells organisms exhibit the capability of acting as a coordinated multi-cellular system. An interesting example of this is bioluminescent bacteria, which coordinate to produce light only when the number of bacteria within a confined region is sufficiently high. The bacteria secrete small diffusible molecules, called autoinducers, which accumulate in the confined space. When there are many bacteria the level of autoinducer in the confined space rises rapidly. The detection of high levels of autoinducer causes bacteria to secrete even more autoinducer and as a result of this positive feedback the level rises exponentially. When the level exceeds some threshold, the bacteria activates its bioluminescent behavior [17].

Quorum sensing is a concept used to describe the ability of cells to determine whether there are sufficient cells (i.e. a quorum) to trigger some activity, such as bioluminescence or virulence. A quorum sensing agent program can be designed based on the bacteria model, which would be useful for situations where it was necessary to determine when the number of agents in a particular state exceeded the minimum threshold for activity.

2.7 Checkpoints and Consensus

Certain decisions require that a set of parallel actions all be completed before proceeding into the next phase. An example of the need for consensus happens during cell division, where the process of dividing can only take place after all chromosome pairs have separated. Such a decision represents a "checkpoint" in the cell cycle; a single failing chromosome pair can halt the process of division until it is successfully separated. The mechanism by which this is achieved is conceptually a very simple distributed process. Each un-separated chromosome pair emits a halting signal and the cell cycle only resumes when all halting signal has disappeared [18].

This simple form of consensus amongst agents can be implemented using active morphogen gradients. Each agent is the source of the halting morphogen, and it discontinues being a source when its internal condition has been satisfied. However the overall morphogen persists until all sources have discontinued. When the last source agent stops producing the morphogen, agents detect the lack of morphogen and can move to the next phase.

Quorum sensing and consensus represent two closely related primitives for achieving coordinated behavior and synchronization, in one case a quorum of agents is sufficient to proceed whereas in the second case all agents must agree to proceed.

2.8 Random Exploration and Selective Stabilization

This last example is another commonly used paradigm at many different levels in biology. The basic concept is: many parallel but random explorations are started and terminated on a regular basis, however occasionally some explorations succeed and these selected explorations stabilize. Furthermore the stabilized explorations may bias the generation of new explorations to occur within its vicinity. Ant foraging behavior is an example of this paradigm. Ants lay random trails in search of food but the trails disappear over time. Successful trails get stabilized by attracting more ants to follow that path and reinforce the scent. Surprisingly, similar processes are observed even within single cells. During cell division, microtubules form a spindle around the chromosomes pairs and then later supply the force to separate them. This spindle is generated by organizing centers that create short-lived microtubules in random directions. However whenever a microtubule attaches to a chromosome, it stabilizes. Eventually all chromosomes become attached and this results in the cell moving on to the next phase of cell division. This general principle is called random exploration and selective stabilization [18].

Agent programs for mobile robots have been designed based on this principle. Such a process however is energy intensive. An alternative method to achieve the same result would be to have the "food" or "chromosomes" create a long-range morphogen gradient and use chemotaxis. While this would be more time and energy efficient, it requires the ability for long-range signaling. This represents an interesting tradeoff between two choices of primitives.

3 Primitives for Robust Global Behavior

In addition to making robust primitives, we also need organizing principles for combining these primitives in ways that produce globally robust results. This area is much less well understood, and the discussion in this section is preliminary. However there are some clear common patterns in development that can guide how we design global languages and compositions of primitives. In this section I will briefly describe five such concepts:

1. Cell differentiation or context-based roles
2. Asynchronous timing and dataflow
3. Compartments and spatial modularity
4. Scale-independence
5. Regeneration

3.1 Cell Differentiation or Context-Based Roles

Rather than start with a completely pre-specified plan for each cell, many embryos first divide into a large number of cells that then slowly differentiate into the various parts of the structure. Often this process takes place as successive rounds of refinement, with coarse regions being determined first and then patterning occurring within those coarse regions. This process has the following interesting property --- cells take on roles based on being *at the right place at the right time*. This simple principle provides a key level of robustness to cell death and cell addition, irrespective of the structure being formed. In many cases cells are capable of changing their fate if conditions change, and this further increases the ability to adapt. At a global programming level this suggests that organizing information as needed and allocating agents when needed, as opposed pre-planning at the level of individual agents, automatically provides a certain degree of robustness, In many of the amorphous computing systems it has been our experience that locally creating information when needed, results in a process that can tolerate small mistakes throughout the system and removes the reliance on any single agent's survival.

3.2 Asynchronous Timing and Dataflow

Embryogenesis involves a cascade of decisions that must take place in a particular sequence. Most embryos show a considerable amount of tolerance to variations in timing and overall development time can be slowed or speeded up without much damage. In a large part, this robustness comes from the fact that the sequence of events does not rely on an absolute global clock [13,11]. Instead the cascade is linked through the flow of information. For example, in the frog embryo the neural plate can not fold into the neural tube before gastrulation (formation of the gut) happens, because it is the process of gastrulation which induces a region of cells to become the neural plate [13]. This form of asynchronous timing is often described as a set of dominos, such that toppling one causes the next to be set in motion. As a result the system can tolerate considerable variation in the timing of individual decisions without significantly affecting the success of the overall cascade. Even more complex forms of timing regulation exist, for example in the wing of the fruit fly different compartments develop independently and may take different amounts of time. However a compartment will wait for the remaining compartments to complete before proceeding to the next phase [15]. One can compare this conceptually to the notion of fork and join in dataflow programming. By using similar notions of asynchronous timing and flow of global information, and by avoiding global clocks, we can design multi-agent systems that can tolerate external causes of timing variations. Most amorphous computing systems use a domino-like timing strategy, and a few actually implement fork and join [7].

3.3 Compartments and Spatial Modularity

A common strategy in embryogenesis is to divide the cells into regions or compartments. An example is the imaginal discs in insects, which are the regions that develop into wings, limbs or antenna [12]. Much of the program for setting up the imaginal disc is common, and abstracted away from the program that generates a specific structure, such as leg. As a result, simple gene manipulations can change the specific structure from one to another, or create additional imaginal discs. This strategy has interesting implications. By capturing common and generic patterns as a spatial module, it leads to a significant reduction in program complexity. It also provides a form of isolation, because cells in two different but non-overlapping compartments can execute the same sequence (make a leg) without interfering. One can think of this as the ability to simultaneously execute the same procedure in different spatial contexts, which provides both program and timing modularity. Regions can be used in multi-agent systems to provide similar benefits, for example morphogens can be confined to propagate within a region thus allowing multiple parts of the system to use the same morphogen name and programs without interference [6].

3.4 Scale-Independence

As mentioned before, many organisms can develop correctly over a large variation in initial embryo size. Frog and sea urchin embryos develop correctly over 8-fold variation and the hydra develops over a few magnitudes of size difference. The ability of an embryo to proportionally regulate with size, without any change in the development program, is called scale-independence or size-invariance [11]. Several models have been proposed for how locally-generated complex patterns could scale.

Wolpert [14] introduced the canonical French Flag problem: write a program for a region of cells such that the cells differentiate into three equally sized sections of blue, white and red, irrespective of the size of the region. One solution he proposed was to use "balancing gradients"; each cell always used the ratio of two morphogens, one from each pole of the region, to determine which part of the global pattern it belonged to. Another method for generating scale-independent patterns has been used in Amorphous Computing, which is to form the pattern by recursively subdividing the space [6]. Each new segmentation decision is taken proportional to the existing set of segmentations, By building in the notion of proportional decisions at the global level of description, one can create complex systems with the ability to adapt to large variations in the number of agents.

3.5 Regeneration

Cockroaches can regenerate amputated limbs in remarkable ways and provide an elegant example of conceptual models of self-repair [11]. If a limb is amputated, the cockroach will regenerate the remaining part. If a section of the limb is removed, it is able to regenerate the missing intervening part. Moreover, if a section of the leg is grafted on backwards, then an entire leg is regenerated in the intervening part. This can be conceptually explained by assuming that there is a gradient of positional information along the leg and that the cells obey the "rule of normal neighbors". If a cell locally detects that its neighbor's positional value is not what it normally expects, it attempts to create a new neighbor with the correct positional value. The new neighbor than repeats this process until all cells are satisfied with their neighbors. This conceptually simple principle applies to regeneration in many different organisms, even though the detailed behavior differs significantly. We have recently used this strategy to successfully create self-repairing patterns and structures in Amorphous Computing. Agents develop a notion of "normal" neighbors, use local monitoring to detect the absence of a neighbor, and react by attempting to recreate/recruit a normal neighbor or committing apoptosis [8.10].

4 Conclusions

In this paper I have presented several examples of biologically-inspired local primitives and global patterns for engineering robust collective behavior. Multi-cellular organisms can provide a metaphor for programming multi-agent systems and for thinking about the kinds of robustness we would like to achieve. However, the goal is not to mimic biology, but to use these evolved systems to help design artificial systems that robustly achieve global goals that we specify. In addition to developing a catalog of basic building blocks one also has to develop the global languages and methods for analysis. Also, different types of goals will require different types of local primitives. For example, development and evolution are two very different processes and solve different problems; whereas development may tell you how to build a fly in spite of the environment, evolution may tell you whether a fly is the appropriate for the environment. Social insects may represent yet another dimension.

As we build multi-agent systems we will encounter many different types of problems and it will be an interesting challenge to know when which strategy is appropriate.

Acknowledgements

This paper discusses many of the projects that were done as part of the Amorphous Computing Project at the Massachusetts Institute of Technology, in Cambridge, MA. The author was supported in part by a grant from NSF on Quantum and Biologically Inspired Computing at MIT.

References

[1] Butler et al.: Generic Decentralized Control for a Class of Self-reconfigurable Robots, International Conference on Robotics and Automation (2002)
[2] Butera, W.: Programming a Paintable Computer. PhD Thesis, Massachusetts Institute of technology (2001)
[3] Payton et al.: Pheromone Robots, Autonomous Robots. vol. 11, no. 3. (2001)
[4] Bojinov et al,.: Multiagent Control of Self-reconfigurable Robots. Intl. Conf. on Multiagent Systems (2000).
[5] Abelson et al.: Amorphous Computing. Communications of the ACM, vol. 43, no. 5, (2000)
[6] Nagpal,: Programmable Self-assembly Using Biologically-inspired Multi-agent Control,. Proc. of Autonomous Agents and Multiagent Systems (2002)
[7] Coore,: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer", PhD Thesis, Massachusetts Institute of Technology (1999)
[8] Kondacs, Biologically-inspired Self-assembly of 2D shapes, Using Global-to-local Compilation", Intl. Joint Conference on Artificial Intelligence (2003)
[9] Nagpal et al.: Programming Methodology for Biologically-inspired Self-assembly. AAAI Spring Symposium (2003)
[10] Clement, Nagpal: Self-Assembly and Self-repairing Topologies. Workshop on Adapatability in Multiagent Systems, Australian Open RoboCup (2003)
[11] Wolpert, L.: Principles of Development. Oxford University Press, UK (1998)
[12] Lawrence, P.: The Making of a Fly. Blackwell Science, Oxford UK (1992)
[13] Slack, J.: From Egg to Embryo. Cambridge University Press, UK (1991)
[14] Wolpert, L.: Positional Information and the Spatial Pattern of Cellular Differentiation, Journal of Theoretical Biology, vol. 25, pages 1-47, (1969)
[15] Day, Lawrence: Morphogens: Measuring Dimensions, the Regulation of Size and Shape. Development 127, (2000)
[16] Lawrence, P.: Morphogens: how big is the big picture?. Nature Cell Biology, vol. 3, (2001)
[17] Miller, M., Bassler, B.: Quorum Sensing in Bacteria. Ann. Rev. Microbiology 55:165-99 (2001)
[18] Alberts et al.: Molecular Biology of the Cell. Garland Publishers, 4th Edition (2002)

Nature-Inspired Self-Organisation in Wireless Communications Networks

Richard Tateson[1], Susan Howard[2], and Ray Bradbeer[2]

[1] BT Exact, Intelligent Systems Lab, Adastral Park, Suffolk, IP5 3RE, UK
richard.tateson@bt.com
[2] QinetiQ, Secure Wireless Solutions Business Group, Malvern Technology Centre
St Andrews Road, Malvern, Worcs. WR14 3PS, UK
{showard2,grbradbeer}@QinetiQ.com

Abstract. An innovative technique for the assignment of frequencies in military combat net radio communications systems is presented. A defined frequency allotment, together with the frequency separations required when stations of different nets are in close physical proximity, gives rise to a complex combinatorial problem, which the assignment algorithms aim to solve. This work seeks an alternative to centralised algorithms, including recently introduced metaheuristics, by using a method based on the self-organising behaviour of cells in a developing fruit fly. Since the method involves a software element representing each net, which interact in a self-organising manner, it has a clear potential for the distribution of the process over a network of processing elements. The demonstrated ability of the algorithm to re-optimise following changes to the network configuration is thought to be a valuable feature in this context, which would be difficult to introduce explicitly using conventional techniques.

1 Introduction

The efficient operation of a radio communications network depends on appropriate assignment of the radio spectrum. The transmission of information requires a finite bandwidth and conventional techniques divide the spectrum into channels of appropriate width to accommodate the transmitted signal [1]. The spectrum is therefore of limited capacity since the transmission of information by that means requires an appropriate element of the spectrum to be sufficiently free of interference to permit satisfactory reception of the signal. Alternative techniques do exist for transmitting many channels of information through an available spectrum, but all are subject to corresponding limitations.

Since radio signals generally diminish in strength with increasing path lengths, radio frequencies can be re-used with appropriate geographic separation. Propagation prediction models can be used to determine anticipated levels of interference and, hence, refine this aspect of the assignment process. The accuracy of this process often depends upon the level of detail with which the communications scenario can be described.

A further major constraint upon the assignment process results from the practical limitations of radio equipment, which mean that significant frequency separation is

© Springer-Verlag Berlin Heidelberg 2004

required if a receiver is to reject successfully the very strong signals emanating from close by transmitters. Other mechanisms give rise to the need to avoid specific combinations of frequencies. Military operational practice requires that communications equipment is able to receive messages in the presence of nearby transmitters, and therefore much attention is given to minimising these problems in the equipment design; however, some remain and must be accommodated in the frequency assignment process.

The result of this analysis is a complex set of frequency separation constraints, giving rise to optimisation, or constraint-satisfaction, problems. According to the precise situation, various forms can be defined. Often it is required to make an assignment using a given set, or allotment, of available channels to obtain a solution which meets all constraints if possible, and minimising violations if not. Other variants of the problem may require minimisation of the span of frequencies used, or the total number of channels employed [2].

The application of metaheuristic techniques to this problem is now well established [3]. Metaheuristics include means to permit the search to escape from local optima. Both simulated annealing and tabu search have successfully been applied to the military frequency assignment problem.

The metaheuristic algorithms aim to minimise the value of a “cost function”, which is derived from the number and magnitude of the constraint violations. Clearly the generation of this function must be designed such that a reduction in the severity of interference is reflected in a reduction in the value of the cost function. Such schemes generally require an initial solution. This can be a random assignment. Faster results can often be obtained by use of a sequential assignment process, which takes the requests in some order, and assesses each available frequency again in some defined order. The first frequency to meet all the constraints, given the assignments already made, is assigned. If no such frequency is available, the request is assigned on a different basis, such as the frequency with minimum cost.

Whatever the details of the optimisation algorithms being used, current approaches share a general philosophy; a high quality solution can be arrived at by a central planner using information about the whole network. This solution can then be implemented in the network, and will remain in place until a review is deemed necessary.

2 The Battlefield Problem

In the military frequency assignment problem, the total network is made up of many communication entities called ‘nets'. Each net consists of many radios communicating on the same channel. Radios on different nets may be located in the same vehicle or within close proximity to one another, thereby causing interference to one another [4].

Information on a military deployment is analysed and a set of complex frequency separation constraints that specify the problem is derived. The frequency separation constraints either specify the number of channels that two nets should be separated by, or exclude the use of a particular frequency by a particular net, as a result of frequencies used by nearby nets. The severity and complexity of the constraints increases the closer two or more radios are to each other.

The most severe of constraints are ‘co-site' constraints, which arise from radios on different nets located in the same vehicle or within close proximity to one another. In a co-site situation, the power ratio between a wanted and an unwanted signal may be $1:10^{14}$. This means that, even with good design, the radio cannot always resolve the wanted signal and reject the unwanted.

Normally several different levels of co-site constraints are employed to reflect the different levels of severity of interference, which arise as a result of different physical separations dictated by operational considerations. Precise values are dictated by the performance of the equipment in use, but are, typically, a few percent of the assigned frequencies.

Co-sited radios additionally suffer from interference given by spurious emissions and intermodulation products. In co-site situations, the large wanted to unwanted signal ratio means that, even with good design, interference can arise because of low level spurious emissions from the transmitter and spurious responses of the receiver. To account for spurious emissions/responses, a net in a co-site is prevented from using a frequency that is a function of the frequency used by a second net in the co-site. The parameters of the function are dependent on the type of radio used on the assigned net. Intermodulation products are generated when the frequencies of two or more radios in a co-site combine to cause potential interference on a different frequency.

Additionally, constraints of only a few channels separation may exist between far-sited radios, i.e. radios on different nets, located in different sites.

3 Frequency Planning in a Dynamic Environment

Traditionally, the military frequency assignment problem is solved by a centralised frequency assignment algorithm using information on the deployment of assets [3, 5, 6].

Centralised techniques work well, but imply the need to collect and update detailed information about the configuration of the system, and subsequently distribute the resulting assignment information. There is potential for a distributed and self-organising system to operate on the basis of distributing information only over such a range as is necessary for the impact of this to be taken into consideration. This could provide a more robust and responsive system.

Such a method differs radically from the centralised approach because no information on the overall behaviour or state of the network is necessary. Potentially this allows the scale of the network to alter without penalty, enabling dynamic solutions which continually re-optimise locally rather than awaiting a network-wide review. Such a system may permit local conditions to be taken into account. For example, interference from other military systems or indeed civil systems might be avoided implicitly, rather than requiring explicit description in the centralised system.

4 The Fruit Fly Inspired Approach

An algorithm inspired by the development of bristles in the larval stage of the fruit fly *Drosophila* has previously been described in [7] and considered in the application to a mobile telephone network.

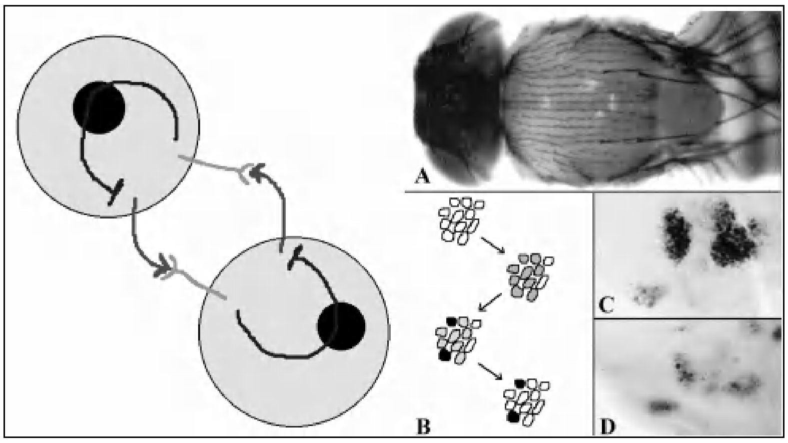


Fig. 1. The diagram on the left shows the mechanism of cell self-organisation. The signal (arrow) is perceived by the receptor and acts to reduce the amount of signal made by the receiving cell. Panel A shows the back of a real fruit fly, with the correct pattern of bristles. B diagrams the progress of a group of cells towards a good pattern: many cells acquire the potential to make bristles (grey shading), of these a few increase bristle-making tendency (black) and inhibit their neighbours (lighter grey and ultimately white). C and D show this process in real tissue. C is the earlier picture, with many cells staining dark for bristle potential. In the later picture (D) most cells are much lighter (less likely to make bristles) but a few are black (certain to make bristles) [10]

The cells in the developing fruit fly must decide whether to become the neural cell-type, which will form the bristles and associated nerves, or the epidermal cell-type, which will form the skin of the fly and the tough exoskeleton through which the bristles protrude. Large groups of cells have the potential to become neural. A mutually inhibitory process operates within each of these groups, with cells synthesising ligand molecules to act as signals to their neighbours, and perceiving the signals from those neighbours using receptor molecules in their cell membranes. The signals are inhibitory - their effect is both to reduce the tendency of a cell to become neural and to reduce the signalling activity of the cell. After a few hours the groups of cells contain only a few cells which have retained their neural potential, amidst a majority of inhibited cells which have been persuaded to adopt the epidermal fate [8, 9].

Self-organisation is a common theme in developmental biology where the freedom from complex information gathering and sophisticated reasoning make the ability to react to local changes to recover correct morphology indispensable. In the case of bristle formation in the fruit fly, disrupting the pattern during the decision-making process, by killing a cell heading for the neural fate, has little effect on the final bristle pattern, since killing the cell also destroys its ability to inhibit its neighbours, one of which will take advantage by becoming a neural cell in place of the dead one.

It has been proposed to use the self-organising behaviour of the fruit fly cells as a model for engineering self-organising behaviour in the base stations of a mobile phone network [7]. The base stations are able to send inhibitory signals directly to their neighbours, and to receive such signals from their neighbours. Instead of cell fate, the base stations are negotiating for the available channels. If the network is to be conFig.d from a naive state, all base stations begin with an almost equal ‘preference' for all channels, with slight ‘noisy' differences. The base stations signal to neighbours and modify their own preferences in the light of incoming signals. Over time, base stations develop highly heterogeneous preferences, falling towards zero on most channels and rising towards maximum on one or a few. These high few are the channels, which will actually be used to carry traffic via that base station.

This algorithm, known as ‘Flyphones', is analogous to the bristle differentiation mechanism in a fruit fly [7]. In the algorithm, each base station acts in parallel, just like the cells in the developing fly. In addition, the presence of a feedback mechanism allows each base station to inhibit its neighbours from using a particular channel. This is similar to the mutually inhibitory process employed by the fruit fly.

At the start of a Flyphones simulation, each base station is assumed to be partially using all of the available channels. Thus, all base stations have almost equal ‘usage' of all channels. Depending on the amount of inhibition that they experience from their neighbours, the base stations will ‘use' some channels more than others as time goes on.

During each successive iteration, the new ‘usage' for each channel of every base station is calculated, based on the current ‘usage' level and the perceived inhibition on the channel in question using equation 1.

$$U_{jkt} = \frac{U_{jk(t-1)}}{1 + I_{jk}} + N \quad (1)$$

where U_{jkt} is the usage of channel k in base station j at time t
I_{jk} is the inhibition calculated for channel k in cell j
N is the noise parameter

At each iteration, it is possible to extract a solution by allocating channels to a base station in descending order of ‘usage' level until the demand for the base station is met.

5 Flyphones for Military Frequency Assignment

It has been previously shown in [7] that the Flyphones algorithm can be successfully applied to the online dynamic channel allocation problem for cellular frequency assignment. This paper looks at extending the Flyphones algorithm to perform dynamic frequency assignment in the challenging environment of military combat radio communications.

Unlike cellular communications, dynamic military frequency assignment carries the restriction that frequency changes to working communications (i.e. nets not currently experiencing interference) should be kept to an absolute minimum. Additionally, the combat radio transceivers are themselves mobile, whereas the base stations of a mobile network are fixed. Thus, the constraints are constantly changing as opposed to changes in the number of channels demanded at each base station.

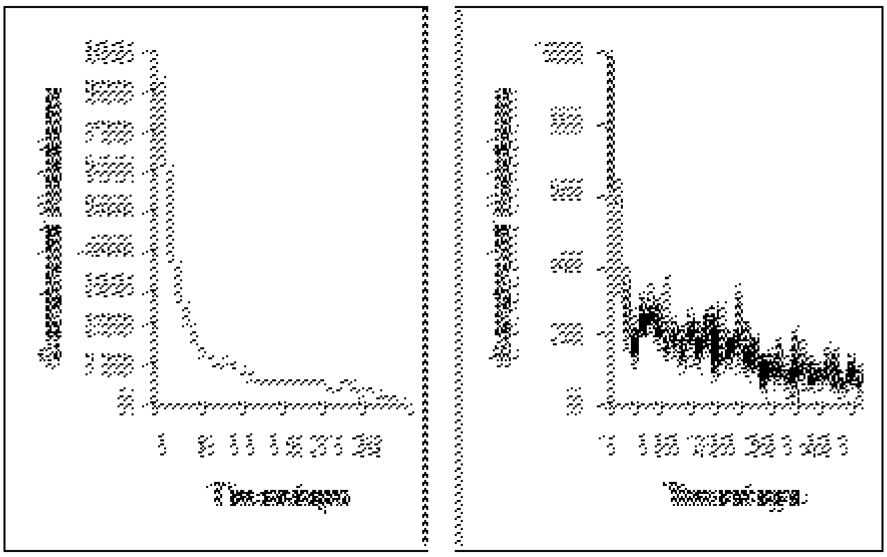


Fig. 2. A single optimisation run, producing a zero cost solution at timestep 471. On the left, the early optimisation phase with rapid reduction of constraint violations. On the right, removing the remaining violations

5.1 First Steps to Dynamic Assignment

Prior to implementing the Flyphones algorithm on the dynamic military assignment problem, it was first necessary to investigate the performance of the algorithm on a static problem and compare it to results obtained using the now established metaheuristic techniques. Testing on a static problem also allowed an investigation into the effects of using different constraint types and the sensitivity of Flyphones to different parameter settings.

The static problem required the assignment of 600 nets, using an allotment of 250 frequencies. Results from centralised metaheuristic optimisers had produced zero constraint-violation solutions including a full consideration of spurious and intermodulation product interference. It was therefore of interest to see whether the adapted Flyphones could also produce ‘perfect' solutions to the problem.

Flyphones was indeed able to solve the problem with zero constraint-violations. An example optimisation run is shown in figure 2. The shape of the curve is typical of Flyphones optimisation for this problem; a steep initial phase in which most of the initial constraint violations (i.e. interference between nets) are removed followed by a long phase attempting to remove the final few violations.

Effects of Varying Noise. The noise parameter in the Flyphones algorithm is the size of the uniform random distribution from which the initial differences between nets, and the perturbations to which nets are subjected in every iteration, are taken. The influence of the noise parameter on the quality of the solutions obtained was tested by varying it in successive optimisation runs. In each instance, the noise parameter is constant across all nets.

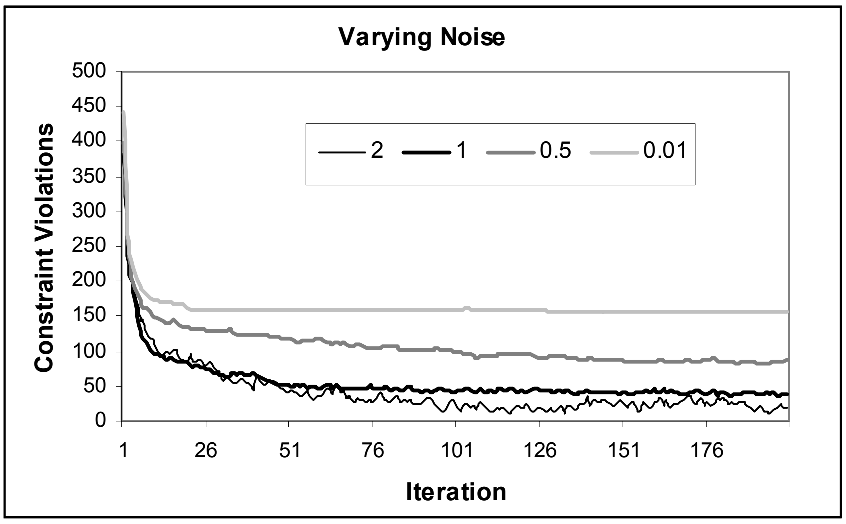


Fig. 3. The number of constraint violations at each time step during four successive runs of the simulation with the noise parameter set to four different values: 2, 1, 0.5 and 0.01

Fig. 3. shows the number of constraint violations at each time step during four successive runs of the simulation. It can be seen that slight increase in the noise level up to a threshold level result in solutions with lower constraint violation. A noise level of 10 is sufficient to destroy the optimisation process and results in random solutions with an average of 1550 constraint violations (not shown).

Comparing Different Types of Frequency Constraints. The time taken to obtain zero cost solutions varies depending on the types of frequency constraints that are considered. Fig. 4. shows the number of constraint violations per unit time during three successive runs of the simulation.

As the time taken to complete a single iteration varies depending on the types of frequency constraints that are considered, the graphs are plotted against time to enable comparisons to be made.

In the first run, only co-site (CS) constraints and far-site (FS) constraints are considered. Next, spurious (SP) constraints are added. Finally, intermodulation (IM) constraints are also considered. Due to the immense amount of calculations involved, the algorithm takes a considerable amount of time to obtain zero cost solutions when all four different types of frequency constraints are considered.

5.2 Dynamic Military Frequency Assignment

As a static optimiser, ‘Flyphones' is able to produce solutions of a similar quality to those produced using the current metaheuristics. However, the real strength of Flyphones is the ability to produce dynamic channel assignment plans.

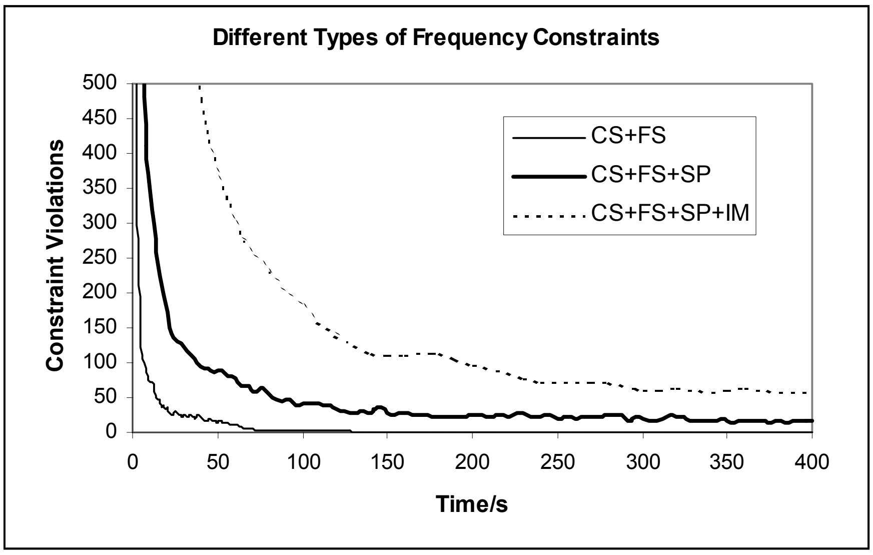


Fig. 4. The difficulty of the static problem increases with additional, more detailed, constraints

To investigate the application of Flyphones for dynamic frequency assignment a second problem was studied. This problem contains 1147 nets, to be assigned from an allotment of 750 frequencies, and considers only far-site and co-site interference. Owing to the lack of spurious and intermodulation constraints, this is a much easier problem in terms of computation of interference and the ease of solution; however, it still captures the essential features and allows us to assess the applicability of Flyphones when confronted with dynamic data.

The dynamic scenario consists of 27 time steps each representing 20-minute intervals over the course of 8 hours and 40 minutes. During each time step movement of nets will occur resulting in changes to the constraints.

Based on the results from the static problem, and some initial runs using the 1147 net problem, the usage and noise parameters were both fixed at 1 for all the dynamic optimisation. This is intended to mimic a situation in which ‘Flyphones' is no longer used as a static optimiser, which is judged on speed of convergence and quality of solution, but as a dynamic frequency management system, which is judged on its ability to maintain viable lines of communication without large disruption to the network. A higher noise setting would allow faster optimisation, but will lead to greater ‘churn' in the network with many nets required to alter frequency.

At time zero (deployment starts) the simulation was initialised with a zero cost assignment (easily found by Flyphones or other standard optimisation methods). The point of interest is to see how Flyphones can adapt that assignment based on the changing constraint data, to minimise interference without needing a complete ‘re-optimisation' requiring all nets to change frequency.

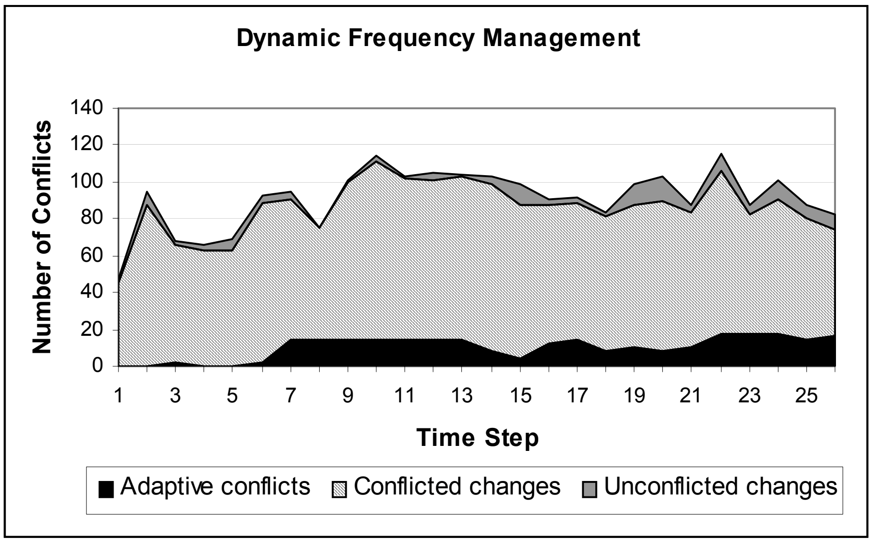


Fig. 5. Dynamic management of combat net radio in simulation by 'Flyphones'. The uppermost line represents the total number of undesirable events in the network. These events are subdivided into three layers. The lowest layer 'adaptive conflicts' shows the number of conflicts remaining after adaptation in each time step. In the middle layer 'conflicted changes' gives the number of nets which had conflicts at the start of the time step, and were required to change frequency to reach the new adapted solution. Finally, the upper layer indicates the number of nets, which did not have any conflicting interference at the start of the time step, but were still required to change frequency

At the start of each time step the solution given at the end of the previous time step is used. The number of nets experiencing conflicts, due to changes in the constraints, is recorded. The Flyphones algorithm adapts the current solution to satisfy the new constraints, whilst trying to cause the least disruption to the current assignment. At the end of the time step the number of nets that have changed frequency is recorded. The number of conflicted changes (those nets that have changed frequency and were previously experiencing interference) and unconflicted changes (those nets that were forced to change frequency even though they were not originally experiencing interference) can be calculated. The number of nets still experiencing interference at the end of the time step (adaptive conflicts) was also recorded.

Fig. 5. shows the results from a typical dynamic optimisation run of this type. At each time step between 50 and 100 conflicted nets were required to change frequency. This means that to adapt the frequency plan to accommodate new far-site and co-site conflicts these nets had to change frequency. This was roughly half of the number of nets that acquired new conflicts at the start of each time step.

The number of constraints that are still violated after adaptation at each time step is not always zero, but it is held below, usually well below, 19 violations. This means that at worst 18 of the 1147 nets still experienced conflicting interference at the end of a time step.

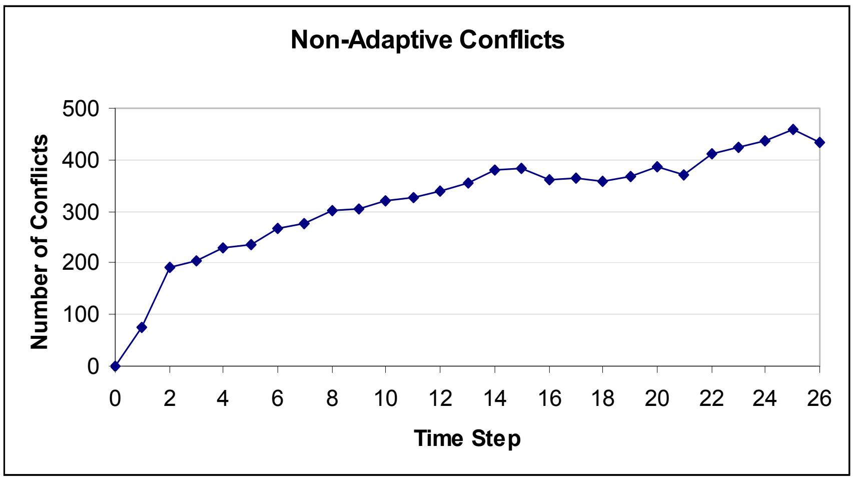


Fig. 6. Degradation of connectivity in the network in the absence of active frequency management

Finally, and in some ways most importantly, the number of nets which were not conflicted, but were still forced to change frequency, is kept low (zero to 13). It would be highly undesirable to require many changes to functioning nets.

In comparison, figure 6 shows the consequences of leaving the original zero cost solution (i.e. the starting point for the dynamic optimisation) in place throughout all time steps without any further optimisation in response to constraint changes. Conflicts accumulate rapidly and lead to roughly one fifth of the network (250 nets) suffering conflicting interference from 2 hours onwards, rising to a third of the network (400 nets) by the end of 8 hours.

6 Conclusions

The algorithm applied in this work has previously been demonstrated to be applicable to the assignment of channels in a mobile radio network. Here it has been shown to be applicable also to the problem of assignment of channels in a military combat radio scenario. Furthermore, it has been demonstrated that the method is comparable to metaheuristic techniques, which have themselves only recently been applied to such problems.

The significant difference between the previously studied mobile radio network and the military radio network studied enhances confidence in the versatility of the method.

Of particular interest in this context, is the demonstrated ability of the algorithm to re-optimise following changes in the frequency assignment constraints arising from changes to the network configuration. In this role, it is possible to alter the balance of the disruption of unconflicted nets against the speed and ultimate effectiveness of the optimisation, by adjustment of the noise parameter. This is thought to be a valuable feature in this context, which would be difficult to introduce explicitly using conventional techniques.

The method is based on the self-organising behaviour of cells in a fruit fly; frequency assignment requests being analogous to the cells in this case. In this work, the method was implemented within a single processor. Since the method involves a software element representing each net requiring assignment, and these interact in a self-organising manner, it has a clear potential for the distribution of the process over a network of processing elements.

The performance, in terms of scaling up and dealing with a dynamic scenario, is extremely encouraging. These results encourage us to believe that there is scope for improving the dynamic management of spectrum. However, the work reported here was done in simulation and there remain issues that need to be addressed if this method is to be brought into use in a real network. The issue of greatest concern is the impact of the negotiation process on the communications capacity of the network.

It has been shown that this self-organising approach to frequency management can readily by applied to a military wireless network simulation. It is anticipated that this algorithm can be adapted for application to areas that are not part of our current research focus, for example wireless local area networks and Bluetooth, both of which operate in a dynamic environment.

Acknowledgements

This research is sponsored by the United Kingdom Ministry of Defence Corporate Research Programme under a Communications Information and Signal Processing project within the Electronic Systems Domain. Many thanks also to Alen Teo for work on the Java simulator and to Derek Smith and Cefn Hoile for helpful advice.

References

[1] Lee, W. C. Y.: Mobile Cellular Telecommunications Systems. McGraw-Hill Book Company, New York (1989).

[2] Smith, D. H., Hurley, S. and Thiel, S. U.: Improving Heuristics for the Frequency Assignment Problem. European Journal of Operational Research, Vol 107. (1998) 76-86.

[3] Smith, D. H., Allen, S. M., Hurley, S. and Watkins, W. J.: Frequency Assignment: Methods and Algorithms. Proceedings of the NATO RTA SET/ISET Symposium on Frequency Assignment, Sharing and Conservation in Systems (Aerospace), Aalborg, Denmark, October 1998, NATO RTO-MP-13, ppK-1 – K-18, (1999).

[4] Bradbeer, R.: Application of New Techniques to Military Frequency Assignment. Proceedings of the NATO RTA SET/ISET Symposium on Frequency Assignment, Sharing and Conservation in Systems (Aerospace), Aalborg, Denmark, October 1998, NATO RTO-MP-13, pp1-1 – 1-5, (1999).

[5] Aarts, E. and Korst, J.: Simulated Annealing and Boltzmann Machines. Wiley (1989).

[6] Lochtie, G. D., van Eijl, C. A. and Mehler, M.J.: Comparison Of Energy Minimising Algorithms For Channel Assignment In Mobile Radio Networks. In: Proceedings of the 8th IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC'97) 3 (1997) 786-790.

[7] Tateson, R.: Self-organising Pattern Formation: Fruit Flies and Cell Phones in A. E. Eiben, T. Back, M. Schoenauer and H-P. Schwefel (eds.): Proc. 5th Int. Conf. Parallel Problem Solving from Nature, Springer , Berlin 732 - 741 (1998).

[8] Artavanis Tsakonas, S., Matsuno, K. and Fortini, M. E.: Notch signalling. Science Vol. 268 (1995) 225-232.

[9] Cubas, P., De Celis, J. F., Campuzano, S. and Modolell, J.: Proneural clusters of achaete-scute expression and the generation of sensory organs in the Drosophila imaginal disc. Genes and Development 5 (6) (1991) 996-1008.

[10] Tateson, R. E.: Studies of the roles of wingless and Notch during the development of the adult peripheral nervous system of Drosophila. PhD Thesis, Cambridge University (1998).

Designing Self-Assembly for 2-Dimensional Building Blocks

Ying Guo, Geoff Poulton, Phil Valencia, and Geoff James

CSIRO ICT Centre,
Marsfield NSW 2122, Australia
{Ying.Guo,Geoff.Poulton,Phil.Valencia,Geoff.James}@csiro.au

Abstract. In this paper we present a genetic algorithm-based approach towards designing self-assembling objects comprised of square smart blocks. Each edge of each block can have one of three polarities (+1, -1 or 0) which defines how blocks stick together - opposite polarities attract, like polarities repel, and a 0 face neither attracts nor repels. In addition to this property, the block contains an internal state machine which can change the polarity of any number of its sides following the detection of an "event" (for example, two blocks sticking or unsticking). The aim of this work is to evolve block parameters and rule sets of the state machine which allow the self-assembly of desired basic structures that can be used as primitive building blocks for the assembly of more complicated objects. We detail a genetic algorithm-based approach that can be used to evolve the rule sets of interaction for a number of interacting blocks, so that the final shape or states of a structure formed by the blocks can approximate some target shapes or satisfy some global goals. We have assumed a list of simple identical properties for each block, and observed that a great diversity of complex structures can be achieved.

1 Introduction

To date, fabrication of small (nano- and molecular-) scale devices has typically been achieved by the meticulous use of precision technologies that are usually not suited for *en-masse* production. Biological systems overcome the problem by using directed self-assembly to produce large-scale systems (including us!) without having to rely on the steady hand of a skilled craftsman. Such natural systems can also usually tolerate (and even correct) errors during growth.

A number of research groups, including CSIRO, have developed means for self-assembly of small (in the order of millimetres to centimetres, sometimes referred to as “meso-scale”) regular lattice-like arrangements as analogues of natural systems [4-7]. Such blocks may be regarded as agents that interact due to their edge properties, which have up to now been regarded as static. The resultant self-assembled object (if one exists) is dependant on the agents' properties such as size, edge polarities, strength of edge fields and many others. However, it should be noted that the environment may also play a crucial role in the object's development [8]. While a large number of objects could be constructed from these static agents by varying their physical parameters, a richer variety of self-assembled objects is possible by allowing

© Springer-Verlag Berlin Heidelberg 2004

the polarity of the edges to change following an “event”, under the control of an internal state machine. In this study an event is defined as a block either sticking to or unsticking from another block (although other choices are possible). For simplicity we assume all blocks to have the same fixed physical shape and internal state machine. This choice can be seen as analogous to discovering the DNA structure and then tweaking the DNA sequence in a biological system. This model is useful in describing the behaviour of self-assembling “intelligent” systems at many scales from macro- to the nano- and molecular scales.

Even for a fixed physical shape and internal state machine there is a huge variety of shapes which can result from the self-assembly process. The unsolved reverse engineering problem is how to specify the block (agent) properties to achieve a prescribed global outcome. The fundamental assumption in this paper is that Genetic Algorithms (GA) can be used to address this reverse engineering problem by evolving these agent properties so that the final structure formed by the agents conforms to the given target shape or global goal.

Of course, even if a general design approach is achieved it is not certain that this will transfer readily to other scales. However, at the least, considerable insight into the processes of self-assembly will be gained.

The physical counterparts to our simulated agents are 2- dimensional blocks floating in fluid, with surfaces coated with self-assembling monolayers which are either hydrophobic (-1), hydrophilic (+1) or neutral (0). By controlling the surface tension and stirring the water, the blocks effortlessly arrange themselves into regular lattice structures [7]. These experiments show that the basic processes of self-assembly operate in a real environment quite similar to that being analyzed.

The remainder of this paper is organized as follows. Section 2 outlines our approach, including the description of the 2-D self-assembly environment and a short introduction of how GA was used. In Section 3, we present preliminary experimental results. Finally, conclusions based on these experiments are discussed in Section 4.

2 The 2D Self-Assembly Environment

As mentioned above, we assume a multi-agent environment where huge number of 2D blocks floating in fluid interact according to their internal properties. These properties are fundamental to the self-assembly process, and are described in detail in the following sections.

2.1 Agent – 2D Block

In many systems, the resultant self-assembled object comprised of a number of basic building blocks will be a direct product of the internal properties of the comprising entities. At the nanoscale, self-assembly control can be achieved by attaching complementary groups to various nanostructures and using the resultant attraction and repulsion properties to coordinate a desired result. We have adopted some of these properties in our 2D block environment with the aim of generating insights into self-assembly processes which may readily be transferred to real world environments.

With this aim in mind we refer to Figure 1 and define the properties of our self-assembly blocks as follows:

(1) Each block has four faces, each of which can have positive, negative, or neutral polarity. This is illustrated in Figure 1, where the four faces of the block are labeled for easy description. The edge state of an agent will be described as a vector of four trinary numbers:

$$\mathbf{Q} = (a_1, a_2, a_3, a_4), where \;\; a_i \in \{+1, -1, 0\} \quad (1)$$

For instance, the edge state of the block in Figure 1 can be described as $\mathbf{Q} = (+1, 0, 0, -1)$. Here +1 stands for positive, -1 for negative, and 0 for neutral.

(2) The internal state machine can change the polarity of each face.

(3) Each block can detect the act of "sticking "or "unsticking".

(4) The internal state machine in each block can use rules based on "sticking" or "unsticking" events to initiate polarity changes on edges.

(5) The initial state of all the blocks in one environment is identical. This includes the edge polarities as well as any internal states.

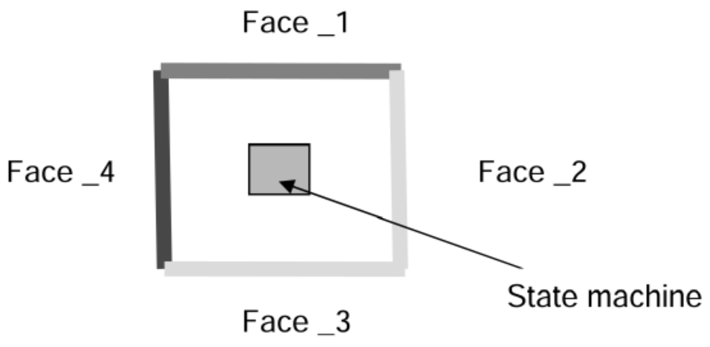


Fig. 1. Structure of each agent in the multi-agent system. Red for positive, blue for negative, and green for neutral

In conjunction with the block properties, the simulation environment attempts to reflect a simplified physical reality in the following manner:

(1) Faces of opposite polarity stick together and generate a "sticking" event to the internal state machines of both "sticking" blocks.

(2) Like polarities repel and will cause connected blocks to "unstick" and subsequently generate an "unsticking" event to the internal state machines of both effected blocks (this however is not utilised in the logic described in this paper).

(3) Neutral polarities will neither stick nor repel any other edges. If a polarity changes to neutral between two connected blocks, an "unstick" will occur and subsequently an "unsticking" event will be communicated to the internal state machines of both effected blocks.

(4) Blocks can rotate within the environment by 90°, 180° or 270°.
(5) For a given situation many different possible structures may self-assemble. However, we will focus only on one randomly chosen structure in the environment. This structure will grow as more blocks stick to it, but may break up due to polarity changes. If a break occurs, we continue to focus only on the further development of the larger of the parts.

Many of these assumptions are based on real-world physical phenomena and are applicable to realizable designs.

Based on these environmental conditions, we can now define a function $A(\mathbf{Q})$ to judge whether one block can stick to its neighbors or will unstick (separate) from the structure.

$$A(\mathbf{Q}) = \sum_{i=1}^{4} p(a_i), \tag{2}$$

where $p(a_i)$ is the power between face i of block $\mathbf{Q}$ and its neighbor's corresponding face:

$$p(a_i) = \begin{cases} +1, \ opposite \ \ polarity \\ -1, \ same \ \ polarity \\ 0, \ one \ \ face \ \ is \ \ neutral \end{cases} \tag{3}$$

Block $\mathbf{Q}$ will stick when $A(\mathbf{Q}) > 0$, or will unstick from other blocks when $A(\mathbf{Q}) \leq 0$. Note that separation functions really need to be defined for all structures, not just for single blocks. In general this is a difficult problem which has not been addressed in the present study.

2.2 Internal States and the Search Space

Let us assume that only a "sticking" event can cause a change in the edge state of a block. In what follows we will investigate the range of changes which are possible under this assumption and the maximum memory which this would require within each block. Since any or all edges of a block may be involved in a sticking event, each such event can change the polarities of all four edges and these changes depend also on the initial edge polarities, there will be many possible combinations. For example, consider the case where only a single edge sticks. Table 1 shows this situation, where S_i is the index of the sticking face, $(a_{1i}, a_{2i}, a_{3i}, a_{4i})$ represents the initial and $(b_{1i}, b_{2i}, b_{3i}, b_{4i})$ the final edge states.

The domain of S_i is {1, 2, 3, 4} whilst that of the a_{ji} and b_{ji} is {-1, 0, +1}. Each line of the table represents a stick event and initial edge state which gives rise to a final edge state. There are $N = 4\times2\times3^3 = 216$ lines (not $4\times3^4 = 324$ since a neutral edge cannot support a sticking event). Each choice of the entire set of b_{ji} represents a possible internal state for the block. Since the b_{ji} are all independently chosen from their domain and there are four per line, the total number of combinations is $3^{4N} = 3^{864}$ which is rather large. In practice this calculation is an overestimate for several reasons, for example if rotated blocks are regarded as identical. This reduces N by a factor of 4 to 54 giving only 3^{216} combinations, a significant reduction although still quite large.

On the other hand only a single sticking event has been allowed in this analysis. If any number of sticks may occur then N increases to 552 (no rotation) or 182 (with rotation) In addition, similar tables may also be defined for “unstick” events, giving even more choice.

Table 1. Possible polarity changes after “sticking"

Stick Face	Initial Edge State	Final Edge State
S_1	a_{11} a_{21} a_{31} a_{41}	b_{11} b_{21} b_{31} b_{41}
S_2	a_{12} a_{22} a_{32} a_{42}	b_{12} b_{22} b_{32} b_{42}
⋮	⋮	⋮
S_N	a_{1N} a_{2N} a_{3N} a_{4N}	b_{1N} b_{2N} b_{3N} b_{4N}

2.2.1 Internal Memory Requirements

Memory requirements are not severe even for the worst case of storing a full table, requiring $4N\log_2 3 = 6.34N$ bits, since only the b_{ji} need to be stored. With all sticks and no rotation about 440 bytes are needed, or 144 bytes if rotation is allowed. As will be seen in later sections full rule tables are rarely needed, in which case memory requirements will be considerably reduced.

2.3 Genetic Algorithm-Based Design

The rule set in the state machine of each agent can be defined in many ways as outlined in Section 2.2. The size of the rule search space can be very large, too big for most optimization techniques to achieve some desired global target. By using a Genetic Algorithm (GA) approach [1], excellent results can be achieved through the evolution of the system. A colony of rule sets can be evolved for a number of generations, improving the performance of the colony. Such techniques are inspired by the processes of natural selection. Techniques of fitness determination, selection, cross-over, reproduction, and mutation are applied to the rules and their chromosomal representation.

At the end of each generation, “parent” rules are selected based on a fitness computation which must be strongly related to the desired outcome. The better the fitness of a given rule, the more likely it is to be selected. After the two parent rules are selected, each is represented by a “chromosomal” string and are then combined, using one of several methods, to form two new chromosomes. These chromosomes are subjected to potential mutation, and are then converted back to their equivalent rule representation. The selected parents are then replaced in the colony by the offspring. The mechanism of natural selection is expected to eventually result in a population with a higher performance.

The first generation (initial population) can either be set randomly, or to predetermined values.

2.3.1 Fitness Functions

After every generated population, every individual of the population must be evaluated to be able to distinguish between good and bad individuals. This is done by mapping the objective function to a "fitness function", which is a non-negative well-behaved measure of relative fitness[1]. The following subsections list some of the fitness functions used in our experiments. All these fitness function definitions lead to maximization problems. Larger fitness values signify rules that have higher fitness.

2.3.1.1 Shape Matching

In this example the goal is for the blocks to self-assemble into a specified target shape. The fitness function is defined to be how closely the final shape matches the desired or target shape, using the formula:

$$f(x) = n_{in} - n_{out} \tag{4}$$

where

n_{in} = the number of filled blocks inside the target shape,

n_{out} = the number of filled blocks outside the target shape.

Of course *f(x)* is defined as the best match under translation and rotation of the structure.

2.3.1.2 Stable Structure

Another important design goal is to generate structures which are stable, that is, will stop growing after reaching a structure for which further sticking events cannot occur. For instance, one cannot stick any new blocks on a structure whose outer faces are all neutral. An appropriate fitness function can thus be defined as:

$$f(x) = \frac{n_{neutral}}{n_{edge}} \tag{5}$$

where

$n_{neutral}$ = the number of outer faces with neutral polarity surrounding the whole structure,

n_{edge} = the total number of outer faces in the structure.

[1] The term "fitness function" is used as a synonym for the "objective function" in this paper. However, the "objective function" is usually desired to be a well-behaved function that maps physical characteristics of the individual to an objective value. This value is then used by the fitness function to determine the fitness of the individual.

Note that in this example many different structures could have identical fitness.

2.3.2 Rule Selection

The selection of the rules for reproduction is based on their performance. The selection process is based on a stochastic universal sampling using roulette wheel selection. The portion of the wheel assigned to a rule is proportional to the rule's performance. Roulette wheel selection thus gives preference to larger portions, as they are more likely to be chosen.

The fitness function used in the genetic algorithm is scaled once the most fit parents have been determined. This fitness scaling is needed in order to prevent a few very fit individual rules from initially taking over a significant proportion of the population, preventing the formation and exploration of new characteristics. Another problem is associated with the later stages of the evolution where the population average fitness and the population best fitness can become too close to one another. In both such cases the quality of the population does not improve rapidly enough. Linear scaling of the fitness function to span the full variation of fitness values can help alleviate these problems:

2.3.3 Rule Combination

The two chromosomes representing the selected rules (parents) must be combined to determine the chromosomes of the two resulting rules (offspring). The combination of the chromosomes is accomplished by first selecting a cross-over point. This is a location within the chromosomal string that divides each chromosome into two parts. The second parts are exchanged to form the offspring chromosomes. In each case there is a small probability that the cross-over operation will not take place. If the cross-over does not happen, the original chromosomes are copied into the next generation.

The next step in the production of new chromosomes for the offspring rules is mutation. Mutation takes place with a certain probability, where a randomly selected bit in the string will be changed from a 0 to a 1, or from a 1 to a 0. Mutation allows the colony to test new rules not resulting from the cross-over operation. The probability of mutation (usually less than 0.1) is generally much lower than that of cross-over (usually greater than 0.5). The values of mutation probability and cross-over probability used in the experiments were determined based on earlier studies.

3 MAS Self-Assembly Experiments

This section addresses the following issues: (1) In which ways do the initial edge states of the agent affect the final structure? (2) What kind of internal states can help us achieve a stable structure? (3) What is the effect of imposing symmetry on the agent?

Three classes of experiments were performed wherein genetic algorithms were utilized to evolve colonies of rules. The first class of experiments, aimed at understanding the effects of the initial states of the agent, was to “grow” desired shapes. Remember that one of our goals is to design the building-blocks that can then be used to “build” more complicated structures. The other two classes of experiment examined the stability of the final structure using equation (5) as the fitness function.

In the self-assembly process, we start with one block in the environment, and then add in one at a time. For each new block, all the stick possibilities are checked with the current structure, and one is chosen at random. The whole assembly process stops once the structure can no longer grow, or has reached a size of M=100 blocks. Because of the random choice, some rule sets can generate many different structures, and achieve different fitness values each time. To evaluate each individual in each generation, the self-assembly process was repeated 4 times, and the fitness value was then averaged over the 4 repeats. In all experiments the rules for the first generation were set randomly. We will show that for some rule sets, the final structure is predictable. For example, the structure can be a desired shape or a stable structure surrounded by neutrals.

3.1 Forward Self-Assembly Simulation – No Internal State Machine

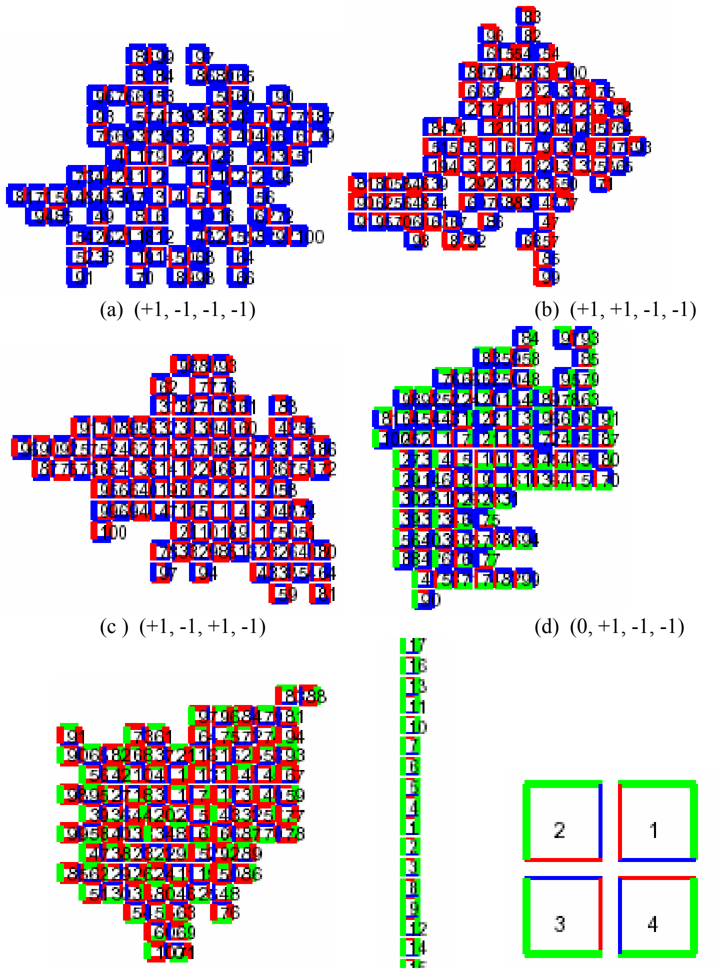
Since it is assumed that all blocks have identical initial states in our experiment environment, it is worth knowing how the initial edge state of the agent affects the final structure. In order to do so, we switched off the state machine in the agent. That is, there is no change after a “stick” event.

Each agent is a square block with four faces, and is assumed to be capable of rotation. Thus, a block with edge state (+1, +1, -1, -1) will be the same as one with (+1, -1, -1, +1). Under these conditions of rotational symmetry we may categorise the possible initial states of an agent in Table 2.

Table 2. Categorised initial states of an agent

Same polarity on four faces		(0,0,0,0), (+1,+1,+1,+1), (-1,-1,-1,–1) the structure cannot grow (no stick event can happen)			
Two polarities on four faces	One is neutral	(+1,0, 0, 0), (-1, 0, 0, 0), (+1,+1,0, 0), … the structure cannot grow (no stick can happen)			
	+1 and –1	(+1,-1,-1,-1),(-1, +1, +1,+1) similar structure Fig. 2.a	(+1,+1,-1,-1) Fig. 2.b	(+1,-1, +1,-1) Fig. 2.c	
Three polarities on four faces		(0,+1,-1,-1), (0, -1, +1,+1),(0,+1,+1,-1), (0, -1, -1, +1) similar structure Figure 2.d	(0, -1, +1,-1), (0,+1,-1, +1) similar structure Figure 2.e	(0,0, +1, -1) Figure 2.f	(+1,0, –1, 0) Figure 2.g

Some structures generated by these initial states are shown in Figure 2. The number in each block is a serial label to record how the structure grows. The structure will stop growing if no more blocks can stick to it, and in this experiment growth was terminated if the number of blocks is more than M. Here $M = 100$.



(a) (+1, -1, -1, -1) (b) (+1, +1, -1, -1)

(c) (+1, -1, +1, -1) (d) (0, +1, -1, -1)

(e) (0, +1,-1,+1) (f) (+1, 0, –1, 0) (g) (0,0,+1,-1)

Fig. 2. Structures generated from different initial block states. The number in each block is a label, which means the order in which this block was added in the structure

This experiment shows that a great diversity of complex structures can be achieved, even though each block's individual behaviour rules are very simple. In Fig. 2.a-f, the structures can keep on growing indefinitely. The only stable structure found may be seen in Figure 2.g when the initial state is $\mathbf{Q} = (0,0,+1,-1)$. For some initial states, e.g. Figure 2.a-e, because every new block will randomly stick to one possible position which satisfies the stick condition, the rule set can generate different structures each time, and the whole process is unpredictable. In other cases regular and predictable structures are generated (see Figure 2.f-g).

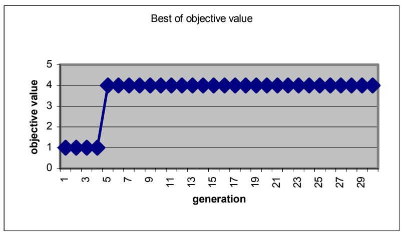
3.2 GA-Based Square-Matching -- No Internal State Machine

Only one stable structure, the 4-block square of Fig. 2.g, was found for blocks with no internal state machine. As an initial trial GA was applied to this simple environment to see if the target shape in Figure 2.g would evolve. The parameters for this experiment are as follows:

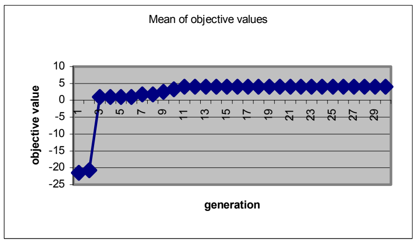
- Variables: the initial state of the block $\mathbf{Q} = (a_1, a_2, a_3, a_4)$.
- Fitness function: the fitness function is defined as in equation (4), where

  target shape = square with four blocks (Fig 2.g).
- A colony with a population of 4 was evolved for 30 generations.

The results are shown in Figure 3. The GA converged very fast, and at least one individual reached the best performance (match to the square shape) after four generations (Fig 3.a). Evolution did create a better colony, where all individuals can generate the square shape after 11 generations (Fig 3.b).



(a) Best individual's fitness value



(b) Mean of fitness value

Fig. 3. GA-Based Square-Matching. (a) The optimum value is 4, which is achieved after 4 generations. (b) After 11 generations, all the individuals can achieve the designed shape – a square of 4 blocks.

3.3 GA-Based Stable-Structure Search - With State Machine

In Sections 2.2 and 2.3 we showed that the size of the rule search space can be huge when the state machine is fully implemented, and hence an algorithm like GA will be necessary to have a chance of achieving desired global targets.

In stead of storing the full rule set in the state machine as set out in Section 2.2 it is possible to make use of just a subset, reducing computational complexity and increasing the likelihood of finding optimum solutions. There are many ways of choosing a useful subset and one method, used in the following experiments, is described as follows.

Given that only a single sticking event is being considered and rotation is not allowed, two further restrictions are put in place: (1) The final state of an edge depends only on its own initial state and which of the four edges sticks; and (2) the state of a stuck face does not change when the stick event happens. This dramatically reduces the number of possibilities. Tables 3 and 4 illustrate this in more detail.

Table 3. Polarity changes after “sticking"

	Due to sticking at face number				
Polarity changes at face number		1	2	3	4
	1	■	▨	▨	▨
	2	▨	■	▨	▨
	3	▨	▨	■	▨
	4	▨	▨	▨	■

Table 4. Possible polarity changes

Off diagonal		On diagonal	
Before	After	Before	After
–1	+1/-1/0	–1	-1
0	+1/-1/0	+1	-1
+1	+1/-1/0		

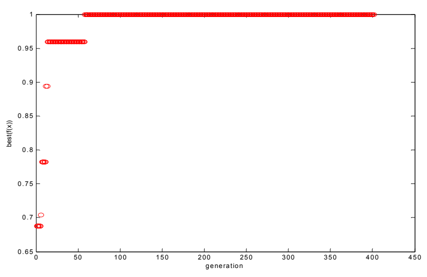
In Table 3, there are 12 shaded squares off the diagonal and 4 dark squares on the diagonal. At each shaded square the polarity changes are chosen from the options given in Table 4. As shown in Section 3.1, the initial states of the block affect the global behaviour, so the initial states of the block $\mathbf{Q} = (a_1, a_2, a_3, a_4)$ should be included as variables. Thus for each block the total number of possible rules, and hence the number of possible system states if the blocks are constrained to be identical, is

$$N_{\text{rules}} = 3^{12\times3+4} = 3^{40}. \tag{6}$$

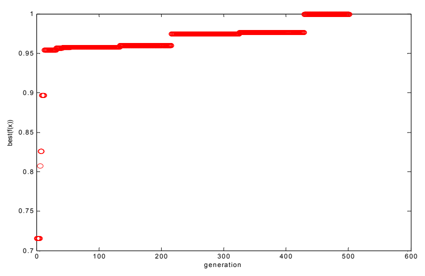
The GA-Based stable-structure searching algorithm is then designed as follows:

- Variables: 40 (see equation (6)).
- Fitness function: Defined as in equation (5).
- Because the size of the rule searching space is large, a colony with a population of 50 was evolved for 400~500 generations.
- An additional constraint was placed on the minimum number of blocks *(nmin)* in the final structure. *nmin* was set to be more than 2, 4, or 6 in three successive experiments.
- The structure will stop growing if no more blocks can stick to it, or the growth will be terminated if the number of blocks is more than M, here taken as 100.

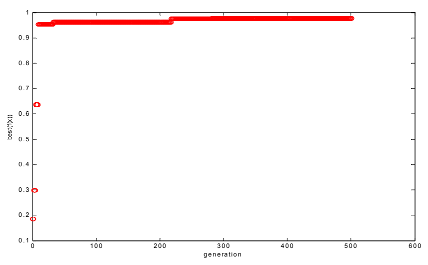
The experimental results are shown in Fig. 4. In Fig. 4.a for *nmin*>2, a stable structure was achieved after about 50 generations. As *nmin* increases it turns out to be harder to find a stable structure. For *nmin*>4 a stable structure was found after about 420 generations (Fig. 4.b), but no stable structure could be found after 500 generations for *nmin*>6, (Figure 4.c). Note that an optimal solution is not guaranteed to be found by GA within 500 generations, so the possibility remains of finding rules to generate stable structures of more than 6 blocks. Some of the final structures achieved based on the rules obtained by GA are shown in Figure 5. The first three are stable, while the last two structures are not, but can still keep on growing in a regular way.



(a) No. of blocks >2



(b) No. of blocks >4



(c) No. of blocks >6

Fig. 4. Best individual's fitness value versus the generation for fitness function (6) and rule table 3. The minimum limit of number of blocks is > 2, 4 or 6 respectively

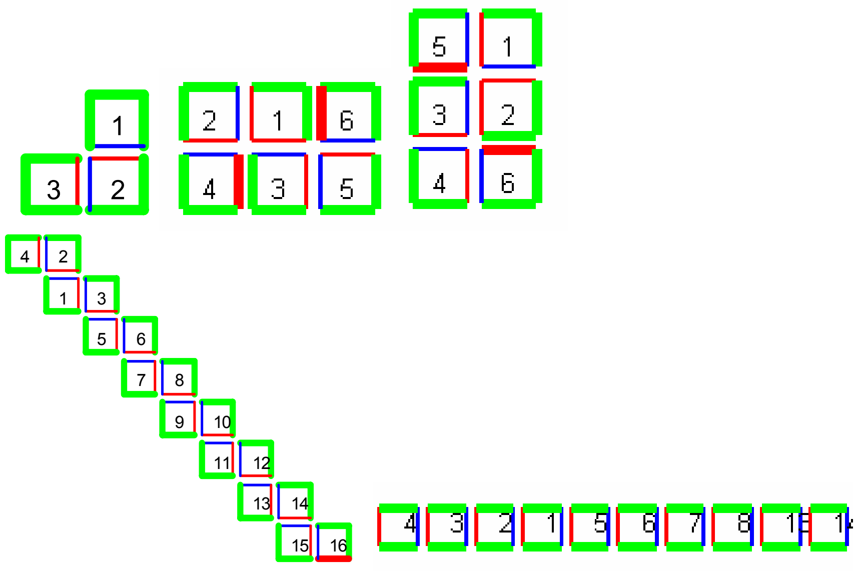
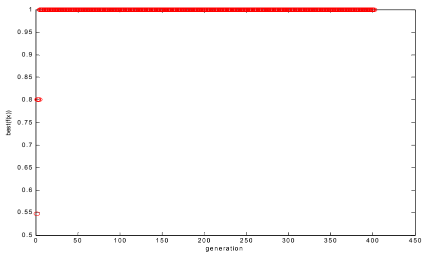


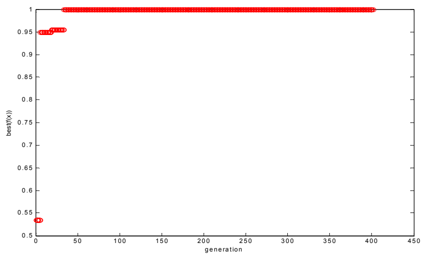
Fig. 5. Some stable or near-stable structures generated by the rules obtained by GA. The last two structures are not stable, but can still keep on growing in a regular way

3.4 Rotated Blocks– Simplified Rule Table

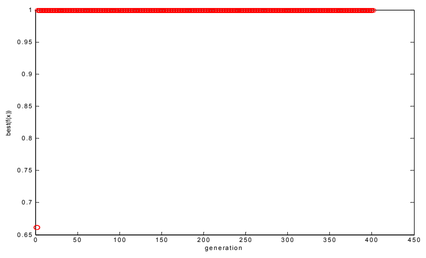
As discussed in Section 2.2 there is a 4-fold redundancy in the rule set if blocks are permitted to rotate For instance, the state $\mathbf{Q}_e$ = (+1 -1 0 0) can generate the same structure as the states (0, 0, +1, -1), (-1, 0, 0, +1), (0, +1, -1, 0). Hence, if we consider the rotation property of 2D blocks and remove such redundancy the corresponding number of rule sets is much reduced, which should improve the performance of GA. This expectation was realized by the experimental results (Fig. 6.), which show that the optimum is reached much sooner than when using the original rule table. In three experiments stable structures were obtained within 40 generations. Fig. 7. shows some of the stable structures based on the experimental results.



(a) No. of blocks >2

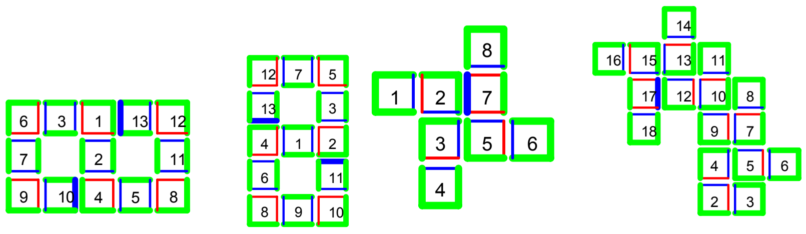


(b) No. of blocks >4



(c) No. of blocks >6

Fig. 6. Best individual's fitness value versus the generation considering the rotation and symmetry properties of agents. The minimum limit of number of blocks is 2, 4 or 6



(a) (b) (c) (d)

Fig. 7. Some structures generated by the rules obtained by GA. One rule can generate different final structures. (a) and (b) are stable structures generated from the same rule set. However, (c) and (d), although coming from the same rule set, appear stable but are not (see 3.4.1)

For the same rule set, the global structure can be different because every sticking event is chosen randomly from all possible sticks. For instance, Figs. 7.a and 7.b are generated from the same rule set and initial block edge state. In this case the initial edge state was (0 -1 +1 -1), and the rule set was

S	a_1	a_2	a_3	a_4	b_1	b_2	b_3	b_4
1	-1	+1	-1	0	-1	0	-1	0
1	1	-1	0	-1	1	1	0	0

Remarkably this rule set is very small, in the sense that only two active rules emerged from the evolutionary process, even though the GA had available to it a huge variety of possible rules (active rules are those that change the state of at least one edge). This parsimonious behaviour is intriguing, but is typical of the experiments carried out in this environment.

3.4.1 Pseudo-stable Structures

The structures shown in Figures 7.c and 7.d appear to be stable but in fact are not. On close examination they can be seen to be composed of multiple structures which, although juxtaposed, are not stuck together. In a real-world environment such structures would float apart. In the current set of experiments it was necessary to select and weed out pseudo-stable structures by hand. In future experiments this should be done automatically.

4 Conclusions and Further Directions

The essential idea of this paper is that a Genetic Algorithm (GA) approach can be used to evolve the rules of interaction for a number of interacting agents, so that the final shape or states of a structure formed by the agents can approximate some target shapes or satisfy some global goals. We have assumed a list of simple identical properties for each agent, and observed that a great diversity of complex structures can be achieved.

It was shown that simulated evolution using genetic algorithms can indeed be used to evolve a colony of rules which conforms to global design goals far better than the initial colony. A list of rule sets was obtained from a huge search space.

We have taken some of the first steps towards the directed self-assembly of 2D mesoblocks, which has relevance ultimately to a wide variety of structured design problems at several scales, particularly when structures are formed by the self-assembly of an agglomeration of intelligent agents. There is much exciting future work to be done in this area, some examples of which are listed below.

- Generate other stop/stable conditions. For instance, each block might have a local clock, and the states of the block will be locked if no event occurs within a certain time.
- Communication between the structure and the environment. Notice that the initial blocks in the experiments are assumed to be the same, which means both positive

and negative properties exist within the initial states in order to generate a structure. This makes the stability of the final structure hard, because the only truly stable condition is if all outside faces of the structure are neutral. If the structure can communicate with the environment, leading to a modification of the states unattached blocks, more interesting stable structures will result.

Acknowledgements

The authors thank Mikhail Prokopenko and Peter Wang for very useful discussions.

References

[1] Goldberg, D.E. Genetic algorithms in search, optimisation, and machine learning. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[2] Wolfram S. ed., Theory and Application of Cellular Automata, World Scientific, Singapore, 1986.

[3] Garis H., Artificial Embryology: The Genetic Programming of an Artificial Embryo, Chapter 14 in book "Dynamic, Genetic, and Chaotic Programming", ed. Soucek B. and the IRIS Group, WILEY, 1992.

[4] Whitesides, G., et al., Mesoscale Self-Assembly of Hexagonal Plates Using Lateral Capillary Forces: Synthesis Using the 'Capillary Bond', J. Am. Chem. Soc., 1999, 121, 5373-5391.

[5] Whitesides, G., et al., Three-Dimensional Mesoscale Self-Assembly, J. Am. Chem. Soc., 1998, 120, 8267-8268.

[6] Whitesides, G., et al., Design and Self-Assembly of Open, Regular, 3D Mesostructures, Science, 1999, 284, 948-951.

[7] Raguse, B., Self-assembly of Hydrogel Mesoblocks, personal communication, CSIRO, 2002.

[8] Bojinov, H., Casal, A., Hogg, T.: Multiagent Control of Self-reconfigurable Robots, Fourth International Conference on MultiAgent Systems, 2000, Boston, Massachusetts.

Strategies for the Increased Robustness of Ant-Based Clustering

Julia Handl, Joshua Knowles, and Marco Dorigo

IRIDIA, Université Libre de Bruxelles
jhandl@iridia.ulb.ac.be
{jknowles,mdorigo}@ulb.ac.be

Abstract. This paper introduces a set of algorithmic modifications that improve the partitioning results obtained with ant-based clustering. Moreover, general parameter settings and a self-adaptation scheme are devised, which afford the algorithm's robust performance across varying data sets. We study the sensitivity of the resulting algorithm with respect to two distinct, and generally important, features of data sets: (i) unequal-sized clusters and (ii) overlapping clusters. Results are compared to those obtained using k-means, one-dimensional self-organising maps, and average-link agglomerative clustering. The impressive capacity of ant-based clustering to automatically identify the number of clusters in the data is additionally underlined by comparing its performance to that of the Gap statistic.

1 Introduction

Ant-based clustering is a heuristic clustering method that draws its inspiration from the behaviour of ants in nature [1]. In particular, it models the clustering and sorting that can be observed in some ant species: where *real ants* gather corpses to clean up their nest, or transport larvae to order them by size, *artificial ants* transport *data items* that are laid out in an artificial environment, and spatially arrange them in a sorted fashion.

Thus, the working principles of ant-based clustering are quite different from those of ordinary clustering algorithms. While well-known clustering methods like k-means or agglomerative clustering gradually build or refine an *explicit* representation of a data set's partitioning, ant-based clustering uses no such model but only implicitly generates the partitioning: all information on the number of clusters and the cluster memberships of individual data items is contained in the final spatial distribution of the data. Also, this outcome is obtained without the explicit specification of an optimisation criterion or global goal, it emerges in a self-organised fashion as the result of local actions and positive feedback only.

As no a priori assumptions on the number, the shape or the size of the clusters in the data need to be made, the risk of producing solutions that are mere artefacts of such assumptions, and which do not reveal any actual information about true data structures, is reduced. In spite of these favourable features,

© Springer-Verlag Berlin Heidelberg 2004

applications of ant-based clustering are rare, as not much is known about the algorithm's real performance and robustness. In this paper we take a step towards a wider application of ant-based clustering by introducing an improved version of the algorithm, and devising generalised parameter settings that permit its application across varying data sets. The algorithm's robustness towards different data properties is then studied using two series of synthetic benchmarks. The obtained results are analysed in terms of the number of clusters identified and in terms of the F-measure; we compare to the number of clusters predicted by the Gap statistic, and to the partitionings generated by each of k-means, average-link agglomerative clustering, and one-dimensional self-organising maps (see Section 4 for details).

The remainder of this paper is structured as follows. Section 2 briefly summarises previous work on ant-based clustering. In Section 3 our new version of the algorithm is described. Section 4 describes the experimental setup, Section 5 discusses results and Section 6 concludes.

2 Ant-Based Clustering

The first ant-based clustering and sorting algorithm was introduced by Deneubourg et al. [2] in 1990, to produce clustering behaviour in simple groups of robots. In the proposed simulation model, ants are modeled as simple agents that randomly move in a square, toroidal environment. The data items that are to be clustered / sorted are initially randomly scattered in this environment and they can be picked up, transported and dropped by the agents. A clustering and sorting of these items is obtained by introducing a bias for the picking and dropping operations, such that data items that are isolated or surrounded by dissimilar ones are likely to be picked up, and transported data items are likely to be dropped in the vicinity of similar ones.

Deneubourg et al. implement this bias by applying the following probabilities for picking and dropping operations:

$$p_{pick}(i) = (\frac{k^+}{k^+ + f(i)})^2$$

$$p_{drop}(i) = (\frac{f(i)}{k^- + f(i)})^2$$

Here, k^+ and k^- are parameters, which determine the influence of the *neighbourhood functions* $f(i)$ and which Deneubourg et al. set to 0.1 and 0.3 respectively. $f(i)$ is an estimation of the fraction of data items in the ant's immediate environment that are similar to the data item i the ant currently considers to pick up / drop. In Deneubourg et al.'s original implementation this estimate is obtained using a short-term memory of each agent, where the contents of the last encountered grid cells are stored, and it therefore only permits the discrimination between a limited number of classes of data items. This limitation has been overcome by Lumer and Faieta [6], who, moving away from the use in robotics, introduced a more general definition of $f(i)$ that permits the algorithm's application to numerical data. An agent deciding whether to manipulate an item i now considers the average similarity of i to all items j in its local neighbourhood:

$$f(i) = \max\left(0, \frac{1}{\sigma^2}\sum_j (1 - \frac{d(i,j)}{\alpha})\right) \tag{1}$$

Here, $d(i,j) \in [0,1]$ is a dissimilarity function defined between points in data space, $\alpha \in [0,1]$ is a data-dependent scaling parameter, and σ^2 is the size of the local neighbourhood (typically, $\sigma^2 \in \{9, 25\}$). The agent is located in the centre of this neighbourhood; its *radius of perception* in each direction is therefore $\frac{\sigma-1}{2}$.

The resulting algorithm still suffers from convergence problems and an unfavourable runtime behaviour, and several attempts to overcome these limitations have therefore been proposed [6, 4].

3 Our Algorithm

In this section, we build upon the work of [2, 6, 4] to develop a general and robust version of ant-based clustering. In particular, we describe how parameter settings for the algorithm can be automatically derived from the data, and we introduce a number of modifications that improve the quality of the clustering solutions generated by the algorithm. In this context, our criteria for 'quality' are twofold: first, in the distribution generated by the ant algorithm on the grid, we desire a clear *spatial separation* between clusters, as this is a requirement both for unambiguously interpreting the solutions and for evaluating them; second, we are interested in a high accuracy of the resulting classification.

Results on the effect of individual modifications are not provided in this paper, but can be found in [3]. Here, we focus on a study of the algorithm's overall performance, in particular its sensitivity with respect to different data properties.

3.1 Basics

The basic ant algorithm (see Algorithm 1) starts with an initialisation phase, in which (i) all data items are randomly scattered on the toroidal grid; (ii) each agent randomly picks up one data item; and (iii) each agent is placed at a random position on the grid. Subsequently, the sorting phase starts: this is a simple loop, in which (i) one agent is randomly selected; (ii) the agent performs a step of a given *stepsize* (in a randomly determined direction) on the grid; and (iii) the agent (probabilistically) decides whether to drop its data item. In the case of a 'drop'-decision, the agent drops the data item at its current grid position (if this grid cell is not occupied by another data item), or in the immediate neighbourhood of it (it locates a nearby free grid cell by means of a random search). It then *immediately* searches for a new data item to pick up. This is done using an index that stores the positions of all 'free' data items on

Algorithm 1 basic_ant

```
begin
INITIALISATION PHASE
Randomly scatter data items on the toroidal grid
for each $j$ in 1 to $\#agents$ do
    $i$ := random_select($remaining\_items$)
    pick_up($agent(j), i$)
    $g$ := random_select($remaining\_empty\_grid\_locations$)
    place_agent($agent(j), g$)
end for
MAIN LOOP
for each $it\_ctr$ in 1 to $\#iterations$ do
    $j$ := random_select($all\_agents$)
    step($agent(j), stepsize$)
    $i$ := carried_item($agent(j)$)
    $drop$ := drop_item?($f^*(i)$) // see equations 3 and 4
    if $drop$ = TRUE then
        while $pick$ = FALSE do
            $i$ := random_select($free\_data\_items$)
            $pick$ := pick_item?($f^*(i)$) // see equations 2 and 4
        end while
    end if
end for
end
```

the grid: the agent randomly selects one data item i out of the index, proceeds to its position on the grid, evaluates the neighbourhood function $f^*(i)$, and (probabilistically) decides whether to pick up the data item. It continues this search until a successful picking operation occurs. Only then the loop is repeated with another agent.

For the picking and dropping decisions the following threshold formulae are used:

$$p^*_{pick}(i) = \begin{cases} 1.0 & \text{if} f^*(i) \le 1.0 \\ \frac{1}{f^*(i)^2} & \text{else} \end{cases} \tag{2}$$

$$p^*_{drop}(i) = \begin{cases} 1.0 & \text{if} f^*(i) \ge 1.0 \\ f^*(i)^4 & \text{else,} \end{cases} \tag{3}$$

where $f^*(i)$ is a modified version of Lumer and Faieta's [6] neighbourhood function (Equation 1):

$$f^*(i) = \begin{cases} \max\left(0, \frac{1}{\sigma^2}\sum_j (1 - \frac{d(i,j)}{\alpha})\right) & \text{if } \forall j \ (1 - \frac{d(i,j)}{\alpha} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

This definition of $f^*(i)$ combines two important properties. First, as in the original neighbourhood function $f(i)$, the division by the neighbourhood size σ^2 penalises empty grid cells, thus inducing a tight clustering (rather than just a loose sorting). Secondly, the additional constraint $\forall j \ (1 - \frac{d(i,j)}{\alpha}) > 0$ serves the purpose of heavily penalising high dissimilarities, which significantly improves spatial separation between clusters.

Note that the above given threshold formulae are quite different from the ones suggested by Deneubourg et al. and are *not* applicable to the basic ant algorithm. They have been experimentally derived for the use with our enhanced version (for which they significantly speed up the clustering process) and have to be seen in light of the shift of the range of attainable values $f^*(i)$ resulting from our increase of the radius of perception (see Section 3.3 below).

3.2 Short-Term Memory

A modified version of the 'short-term memory' introduced by Lumer and Faieta in [6] is employed. In their approach, each agent remembers the last few carried data items and their respective dropping positions. When a new data item is picked up, the position of the 'best matching' memorised data item is used to bias the direction of the agent's random walk. Here, the 'best matching' item is the one of *minimal dissimilarity* $d(i,j)$ to the currently carried data item i. We have extended this idea as follows.

In a multi-agent system the items stored in the memory might already have been removed from the remembered position. In order to determine more robustly the direction of bias, we therefore permit each ant to exploit its memory as follows: An ant situated at grid cell p, and carrying a data item i, uses its memory to proceed to all remembered positions, one after the other. Each of them is evaluated using the neighbourhood function $f^*(i)$, that is, the suitability of each of them as a dropping site for the currently carried data item i is examined. Subsequently, the ant returns to its starting point p.

Out of all evaluated positions, the one of 'best match' is the grid cell for which the *neighbourhood function yields the highest value*. For the following step of the ant on the grid, we replace the use of a biased random walk with an agent 'jump' directly to the position of 'best match'. However, this jump is only made with some probability, dependent on the quality of the match; the same probability threshold that we use for a dropping operation $p^*_{drop}(i)$ is used for this purpose. If the jump is not made, the agent's memory is de-activated, and in future iterations it reverts to trying random dropping positions until it successfully drops the item.

3.3 Increasing Radius of Perception

The size of the local neighbourhood perceived by the ants limits the information used during the sorting process. It is therefore attractive to employ larger neighbourhoods in order to improve the quality of the clustering and sorting on the grid. However, the use of a larger neighbourhood is not only more expensive (as the number of cells to be considered for each action grows quadratically with the radius of perception), but it also inhibits the quick formation of clusters during the initial sorting phase.

We therefore use a radius of perception that gradually increases over time. This saves computations in the first stage of the clustering process and prevents difficulties with the initial cluster formation. At the same time it accelerates the dissolution of preliminary small clusters, a problem that has already been addressed in [6, 4]. In the current implementation, we start with an initial perceptive radius of 1 and linearly increase it to be 5 in the end. While doing so, we leave the scaling parameter $\frac{1}{\sigma^2}$ in Equation 4 unchanged, as its increase results in a loss of spatial separation.

This brings about the gradual shift in the range of attainable values $f^*(i)$ that we have mentioned in Section 3.1. In the starting phase of the algorithm, $f^*(i)$ is limited to the interval $[0, 1]$; the upper bound, however, increases with each increment of the neighbourhood radius, such that, in our implementation, $f^*(i)$ can yield values within the interval $[0, 15]$ after the last increment. Consequently, the picking operation is purely deterministic in the beginning, and, at this stage, it is the dropping operation solely that favours dense and similar neighbourhoods. Gradually, with the rise of $f^*(i)$, an additional bias towards the picking of misplaced data items is introduced. The shift of the values of $f^*(i)$ combined with the use of the threshold functions (Equations 2 and 3) has the effect of decreasing the impact of density for the dropping threshold while, simultaneously, increasing it for the picking threshold. This results in an improved spatial separation between clusters.

3.4 Spatial Separation

As stated above, the spatial separation of clusters on the grid is crucial in order for individual clusters to be well-defined. Spatial closeness, when it occurs, is, to a large degree, an artefact of early cluster formation. This is because, early on in a run, clusters will tend to form wherever there are locally dense regions of similar data items; and thereafter these clusters tend to drift only very slowly on the grid. After an initial clustering phase, we therefore use a short interlude (from time t_{start} to t_{end}) with a modified neighbourhood function, which replaces the scaling parameter $\frac{1}{\sigma^2}$ by $\frac{1}{N_{occ}}$ in Equation 4, where N_{occ} is the *actual observed number* of *occupied* grid cells within the local neighbourhood. Hence only similarity, not density, is taken into account, which has the effect of spreading out data items on the grid again, but in a sorted fashion; the data items belonging to different clusters will now occupy individual 'regions' on the grid. Subsequently, we turn back to using the traditional neighbourhood function. Once again, clear clusters are formed, but they now have a high likelihood of being generated along the centres of these 'regions', due to the lower neighbourhood quality at their boundaries.

3.5 Parameter Settings

Ant-based clustering requires a number of different parameters to be set, some of which have been experimentally observed to be independent of the data. These include the number of agents, which we set to be 10, the size of the agents' short-term memory, which we equally set to 10, and t_{start} and t_{end}, which we set to $0.45 \cdot \#iterations$ and $0.55 \cdot \#iterations$.

Parameters to Be Set as a Function of the Size of the Data Set. Several other parameters should however be selected in dependence of the size of the data set tackled, as they otherwise impair convergence speed. Given a set of N_{items} items, the grid (comprising a total of N_{cells} cells) should offer a sufficient amount of 'free' space to permit the quick dropping of data items (note that each grid cell can only be occupied by one data item). This can be achieved by keeping the ratio $r_{occupied} = \frac{N_{items}}{N_{cells}}$ constant and sufficiently low. A good value, found experimentally, is $r_{occupied} = \frac{1}{10}$. We obtain this by using a square grid with a resolution of $\sqrt{10N_{items}} \times \sqrt{10N_{items}}$ grid cells. The stepsize should permit sampling of each possible grid position within one move, which is obtained by setting it to $stepsize = \sqrt{20N_{items}}$. The total number of iterations has to grow with the size of the data set. Linear growth proves to be sufficient, as this keeps the average number of times each grid cell is visited constant. Here, $\#iterations = 2000 \cdot N_{items}$, with a minimal number of 1 million iterations imposed.

Activity-Based α-Adaptation. An issue already addressed in [4] is the automatic determination of the parameter α (recall that α is the parameter scaling the dissimilarities within the neighbourhood function $f^*(i)$), which the functioning of the algorithm crucially depends on. During the sorting process, α determines the percentage of data items on the grid that are classified as similar, such that: a too small choice of α prevents the formation of clusters on the grid; on the other hand, a too large choice of α results in the fusion of individual clusters, and in the limit, all data items would be gathered within one cluster.

Unfortunately, a suitable choice of the parameter α depends on the distribution of pairwise dissimilarities within the collection and, hence, cannot be fixed without regard to the data. However, a mismatch of α is reflected by an excessive or extremely low sorting activity on the grid. Therefore, an automatic adaptation of α can be obtained through the tracking of the amount of activity, which is reflected by the frequency of the agents' successful picking and dropping operations. The scheme for α-adaptation used in our experiments is described below.

A heterogenous population of agents is used, that is, each agent makes use of its own parameter α. All agents start with an α parameter randomly selected from the interval [0, 1]. An agent considers an adaptation of its own parameter after it has performed N_{active} moves. During this time, it keeps track of the failed dropping operations N_{fail}. The rate of failure is determined as $r_{fail} = \frac{N_{fail}}{N_{active}}$ where N_{active} is fixed to 100. The agent's individual parameter α is then updated using the rule

$$\alpha \leftarrow \begin{cases} \alpha + 0.01 \text{ if } r_{fail} > 0.99 \\ \alpha - 0.01 \text{ if } r_{fail} \leq 0.99 \end{cases}$$

which has been experimentally derived. α is kept adaptive during the entire sorting process. This makes the approach more robust than an adaptation method with a fixed stopping criterion. Also, it permits for the specific adaptation of α within different phases of the sorting process.

4 Evaluation

In the following we briefly describe the main experimental setup used for the evaluation of the described algorithm.

Comparison. The performance of a clustering algorithm can best be judged with respect to its relative performance when compared to other algorithms. We therefore choose three popular clustering methods from the literature, the partitioning method *k-means* [7], the hierarchical clustering algorithm *average-link agglomerative clustering* [12], and *one-dimensional self-organising maps* (1D-SOM, [5]).

All three of these algorithms require the correct number of clusters as an input parameter. While automatic and semi-automatic methods for the determination of the number of clusters within a data set exist (cf. [8] for a survey), none of these is infallible. In order to avoid the introduction of an additional source of error we therefore provide the correct number of clusters to k-means, average link agglomerative clustering and 1D-SOM, thus giving the same advantage to all three algorithms.

Other implementation details are as follows: The standard version of average-link agglomerative clustering is used. k-means is implemented using batch training and random initialisation, and only the best result out of 20 runs (in terms of the minimal intra-cluster variance) is returned. 1D-SOM is implemented in accordance with the description given in [11]: Sequential training, uniform initialisation of the weight vectors, a rectangular grid and a 'bubble' neighbourhood are used. The training consists of two phases: a first 'coarse' approximation phase of 10 iterations, with a learning rate of $lr = 0.5$ and the neighbourhood size decreasing exponentially from $\max(1.0, \frac{k}{4})$ to $\max(1.0, \frac{k}{16})$, and a second fine-tuning phase of 40 iterations, with a learning rate of $lr = 0.05$ and the neighbourhood size decreasing exponentially from $\max(1.0, \frac{k}{16})$ to 1.0 (here, k is again the number of clusters). In each iteration all data elements are presented to the SOM.

For the evaluation of ant-based clustering's performance at identifying the correct number of clusters in the data, we additionally compare against the results returned by the Gap statistic, a recently proposed automated method for the determination of the number of clusters in a data set [9]. This statistic is based on the expectation that the most suitable number of clusters appears as a significant 'knee' in a plot of the performance of a clustering algorithm versus the number of clusters, k. For this purpose, the clustering problem is solved for a range of different values of k and, for each k, the resulting partitioning

$C = \{C_1, .., C_k\}$ is evaluated by means of the intra-cluster variance, which is given by

$$V(k) = \sum_{C_i \in C} \sum_{j \in C_i} (d(j, \mu_i))^2.$$

Here C_i is the ith cluster in the partitioning, μ_i is the corresponding cluster centre, and $d(j, \mu_i)$ gives the dissimilarity between data item j and μ_i. The intra-cluster variance is affected by the number of clusters, such that a plot of $V(k)$ exhibits a decreasing trend that is *solely* caused by the finer partitioning and *not* by the actual capturing of structure within the data. The Gap statistic overcomes this effect through a normalisation of the performance curve. B reference curves $R_b(k)$ (with $b \in \{1, ...B\}$) are computed, which are the performance curves obtained with the same clustering algorithm for uniform random reference distributions. Using these, the normalised performance curve ('Gap curve') for $V(k)$ is then given as

$$Gap(k) = \frac{1}{B} \sum_{b=1}^{B} log(R_b(k)) - log(V(k)).$$

The most suitable number of clusters is determined by finding the first significant local maximum of $Gap(k)$.

For our implementation of the Gap statistic we use the above described k-means algorithm. We compute the performance curves for $k \in \{1, ..., 20\}$, and, for each k, we generate $B = 20$ reference distributions.

Benchmark Data. The benchmarks used in our experiments are synthetic data sets with each cluster generated by a two-dimensional normal distribution $N(\boldsymbol{\mu}, \boldsymbol{\sigma})$. The number of clusters, the sizes of the individual clusters, and the mean vector $\boldsymbol{\mu}$ and vector of the standard deviation $\boldsymbol{\sigma}$, for each normal distribution, are manually fixed. In the experiments, each algorithm is run 50 times on each type of benchmark, and for every individual run, the actual data items are newly sampled from the normal distributions.

Table 1 gives the definition of the benchmarks, and Figure 1 shows four sample instances. The benchmarks are variations of the *Square* data set, a data set that has been frequently employed in the literature on ant-based clustering. It is two-dimensional and consists of four clusters of equal size (250 data items each), which are generated by normal distributions with a standard deviation of 2 in both dimensions and are arranged in a square.

The data sets *Square1* to *Square7* only differ by the distance between the individual clusters (i.e., the length of the edges of the square), which is 10, 9, 8, 7, 6, 5 and 4 respectively. They were generated in order to study the relative sensitivity of the algorithms to increasing overlap between clusters.

In the *Sizes1* to *Sizes5* data sets, edge length and standard deviation are kept constant, and, instead, they differ in the sizes of the individual clusters. In particular, the ratio between the smallest and the largest cluster increases from

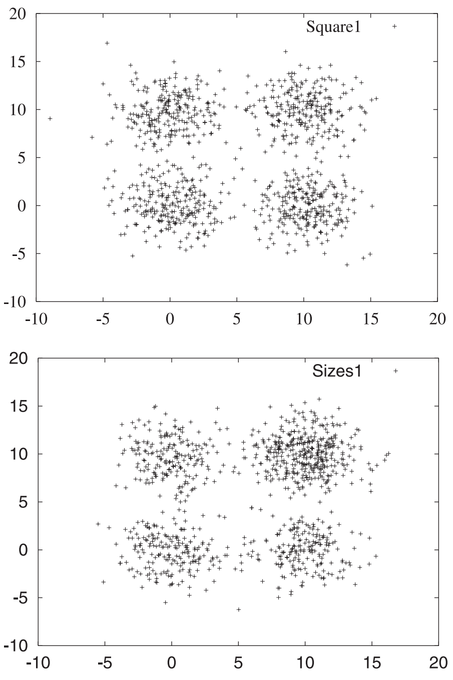


Fig. 1. Four sample instances of the *Square1*, the *Square5*, the *Sizes1* and the *Sizes5* benchmark data

the *Sizes1* (where it is 2) to the *Sizes5* data (where it is 10). This is used to investigate the algorithms' sensitivity to unequally-sized clusters.

Data Preprocessing. Prior to clustering, the data is normalised in each dimension. The Euclidean distance is used as a distance measure between individual data items, and, for average-link agglomerative clustering and ant-based clustering the complete dissimilarity matrix is precomputed.[1] The entries of the matrix are normalised to lie within the interval $[0, 1]$.

Analytical Evaluation. The analytical evaluation of the performance of ant-based clustering requires that the solution generated by the algorithm be made explicit, that is, that the spatial distribution produced by the algorithm be converted to an explicit partitioning of the data. In our experiments this was done using the automated method described in [3].

We evaluate the obtained partitioning using the F-Measure [10], which combines information on the purity and the completeness of the generated clusters

[1] Note that this is not possible for k-means and 1D-SOM, as these work with cluster representatives that do not necessarily correspond to actual data items within the collection and can change in each iteration.

Table 1. Summary of the used data sets. *dim* is the dimensionality, k gives the number of clusters, and n_j gives the number of data elements for cluster C_j. The test sets are generated by multidimensional normal distributions $N(\boldsymbol{\mu}, \boldsymbol{\sigma})$, where $\boldsymbol{\mu}$ is the vector of means and $\boldsymbol{\sigma}$ is the vector of the standard deviations

Name	k	n_j	dim	Source
Square1	4	4×250	2	$N([0,0],[2,2]),\ N([10,10],[2,2])$ $N([0,10],[2,2]),\ N([10,0],[2,2])$
Square2	4	4×250	2	$N([0,0],[2,2]),\ N([9,9],[2,2])$ $N([0,9],[2,2]),\ N([9,0],[2,2])$
Square3	4	4×250	2	$N([0,0],[2,2]),\ N([8,8],[2,2])$ $N([0,8],[2,2]),\ N([8,0],[2,2])$
Square4	4	4×250	2	$N([0,0],[2,2]),\ N([7,7],[2,2])$ $N([0,7],[2,2]),\ N([7,0],[2,2])$
Square5	4	4×250	2	$N([0,0],[2,2]),\ N([6,6],[2,2])$ $N([0,6],[2,2]),\ N([6,0],[2,2])$
Square6	4	4×250	2	$N([0,0],[2,2]),\ N([5,5],[2,2])$ $N([0,5],[2,2]),\ N([5,0],[2,2])$
Square7	4	4×250	2	$N([0,0],[2,2]),\ N([4,4],[2,2])$ $N([0,4],[2,2]),\ N([4,0],[2,2])$
Sizes1	4	$400, 200, 200, 200$	2	$N([0,0],[2,2]),\ N([10,10],[2,2])$ $N([0,10],[2,2]),\ N([10,0],[2,2])$
Sizes2	4	$571, 143, 143, 143$	2	$N([0,0],[2,2]),\ N([10,10],[2,2])$ $N([0,10],[2,2]),\ N([10,0],[2,2])$
Sizes3	4	$667, 111, 111, 111$	2	$N([0,0],[2,2]),\ N([10,10],[2,2])$ $N([0,10],[2,2]),\ N([10,0],[2,2])$
Sizes4	4	$727, 91, 91, 91$	2	$N([0,0],[2,2]),\ N([10,10],[2,2])$ $N([0,10],[2,2]),\ N([10,0],[2,2])$
Sizes5	4	$769, 77, 77, 77$	2	$N([0,0],[2,2]),\ N([10,10],[2,2])$ $N([0,10],[2,2]),\ N([10,0],[2,2])$

with respect to the real class memberships. In particular, it adopts the ideas of precision and recall from information retrieval: Each *class* T_i (inherent to the data) is regarded as the set of n_i items desired for a query; each *cluster* C_j (generated by the algorithm) is regarded as the set of n_j items retrieved for a query; n_{ij} gives the number of elements of class T_i within cluster C_j. For each class T_i and cluster C_j precision and recall are then defined as $p(i,j) = \frac{n_{ij}}{n_j}$ and

$r(i,j) = \frac{n_{ij}}{n_i}$, respectively, and the corresponding value under the F-Measure is

$$F(i,j) = \frac{(b^2+1) \cdot p(i,j) \cdot r(i,j)}{b^2 \cdot p(i,j) + r(i,j)},$$

where equal weighting for $p(i,j)$ and $r(i,j)$ is obtained if $b = 1$. The overall F-value for the partitioning is computed as

$$F = \sum_i \frac{n_i}{n} \max_j \{F(i,j)\}.$$

It is limited to the interval $[0,1]$ and should be maximised.

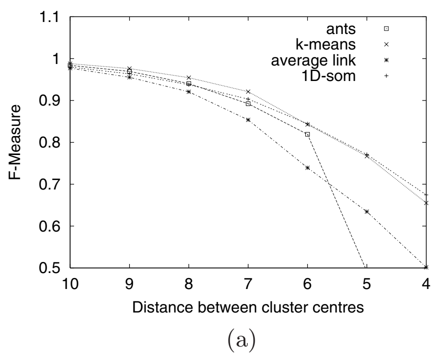
5 Results

We now summarise the results obtained in our comparison of the ant-based clustering algorithm with the standard clustering techniques k-means, average-link agglomerative clustering and 1D-SOM. While, in this paper, we limit the discussion to the qualitative performance of ant-based clustering on the two types of synthetic data presented in the above section, results for more general synthetic and real data sets (including runtimes) can be found in [3].

Sensitivity to Overlapping Clusters. We study the sensitivity to overlapping clusters using the *Square1* to *Square7* data sets. It is clear that the performance of all four algorithms necessarily has to decrease with a shrinking distance between the clusters, as points within the region of overlap cannot be correctly classified. It is however interesting to see whether the performance of the individual algorithms degrades gracefully or more catastrophically, as a graceful degradation would indicate that the main cluster structures are still correctly identified.

Figure 2a shows a plot of the algorithms' performance (as reflected by the F-measure) versus the distance between neighbouring clusters. A number of trends can be observed in this graph. There is the very strong performance of k-means, which performs best on the first four data sets. The 1D-SOM starts on a lower quality level, but its relative drop in performance is less than that of k-means: it clearly profits from its topology preserving behaviour, which makes it less susceptible to noise. Average-link agglomerative clustering, in contrast, has trouble in identifying the principal clusters and performs quite badly, especially on the data sets with a lower inter-cluster distance.

The results of ant-based clustering are very close to those for k-means on the simplest data set, *Square1*, but its performance drops slightly more quickly. Still, it performs significantly better than average-link agglomerative clustering on the first five test sets. Also, in spite of the fact that the clusters 'touch', the ant-algorithm reliably identifies the correct number of clusters on the first five test sets, and it can thus be concluded that the algorithm does not rely on



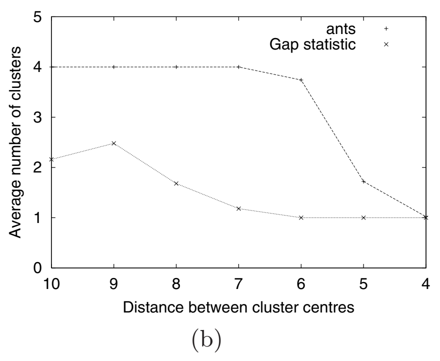


Fig. 2. Performance as a function of the distance between the cluster centres on the *Square* data sets. (a) F-Measure (b) Number of identified clusters

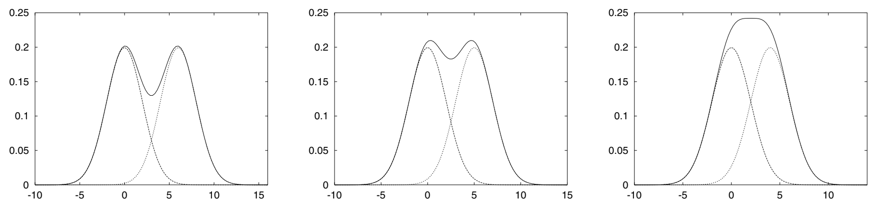


Fig. 3. Theoretical density distribution along the connecting line between two cluster centres in the *Square5*, the *Square6* and the *Square7* test set (from left to right). Constructed as the superimposition of two one-dimensional normal distributions with standard deviation 2 and a distance of 6, 5 and 4 respectively

the *spatial separation* between clusters, but that distinct changes in the *density distribution* are sufficient for it to detect the clusters.

For the *Square6* and *Square7* test data, the performance of ant-based clustering drops significantly, as it fails to reliably detect the four clusters. For the *Square6* test set the number of identified clusters varies between 1 and 4, for the *Square7* only 1 cluster is identified (see Figure 2b). However, a plot of the theoretical density distribution along the 'edge' between two neighbouring clusters in this data set, puts this failure into perspective: Figure 3 makes clear, that, due to the closeness of the clusters, the density gradient is very weak for the *Square6* data, and the distribution of data items is nearly uniform for the *Square7* data.

The reader should keep in mind that, different from its competitors, ant-based clustering has not been *provided* with the correct number of clusters. In order to get a more precise idea of the performance of ant-based clustering, we therefore additionally analyse its success at identifying the correct number of clusters in the data. The comparison in Figure 2b shows that ant-based clustering performs very well, it is much less affected by the lack of spatial separation between the clusters than the Gap statistic.

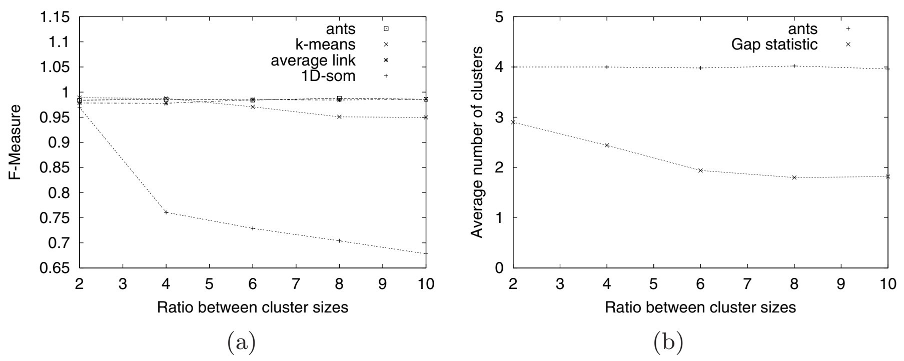


Fig. 4. Performance as a function of the ratio between cluster sizes. (a) F-Measure (b) Number of identified clusters

Sensitivity to Unequally-Sized Clusters. The sensitivity to unequally-sized clusters is studied using the *Sizes1* to *Sizes5* data sets. Again, we show the algorithms' performance on these data sets as reflected by the F-Measure (Figure 4).

Ant-based clustering performs very well on all five test sets, in fact it is hardly affected at all by the increasing deviations between cluster sizes. Out of its three contestants, only average-link agglomerative clustering performs similarly robustly. 1D-SOM is very strongly affected by the increase of the ratio between cluster sizes, and the performance of *k*-means also suffers. The performance of the Gap statistic is again very weak when compared to ant-based clustering.

6 Conclusion

In this paper we have introduced algorithmic modifications and parameter settings for ant-based clustering that permit its direct application to arbitrary numerical data sets. While the robust performance of the algorithm across a wide range of test data has been demonstrated elsewhere [3], our analysis in this paper has focused on studying two particular data properties that can pose problems to clustering algorithms.

The results presented demonstrate the robust performance of ant-based clustering. The algorithm is largely unaffected by data sets in which the clusters are unequally sized, and it succeeds at reliably separating clusters up to a high degree of overlap. In both cases, it clearly outperforms the Gap statistic at identifying the correct number of clusters.

Acknowledgements

Julia Handl is supported by a DAAD scholarship, and thanks Prof. Günther Görz, Universität Erlangen-Nürnberg, for his supervision. Joshua Knowles gratefully acknowledges the support of a CEC Marie Curie Fellowship, contract:

HPMF-CT-2000-00992 (to September 2003) and a BBSRC David Phillips Fellowship (from October 2003). Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Senior Research Associate, and from the "ANTS" project, an "Action de Recherche Concertée" funded by the Scientific Research Directorate of the French Community of Belgium.

References

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence – From Natural to Artificial Systems*. Oxford University Press, New York, NY, 1999. 90

[2] J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. The dynamics of collective sorting: Robot-like ants and ant-like robots. In J.-A. Meyer and S. Wilson, editors, *Proceedings of the First International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 1*, pages 356–365. MIT Press, Cambridge, MA, 1991. 91, 92

[3] J. Handl. Ant-based methods for tasks of clustering and topographic mapping: extensions, analysis and comparison with alternative methods. Masters thesis. Chair of Artificial Intelligence, University of Erlangen-Nuremberg, Germany. November 2003. `http://www.handl.julia.de`. 92, 99, 101, 103

[4] J. Handl and B. Meyer. Improved ant-based clustering and sorting in a document retrieval interface. In *Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature*, volume 2439 of *LNCS*, pages 913–923. Springer-Verlag, Berlin, Germany, 2002. 92, 95, 96

[5] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, Germany, 1995. 97

[6] E. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In *Proceedings of the Third International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 3*, pages 501–508. MIT Press, Cambridge, MA, 1994. 91, 92, 93, 94, 95

[7] L. MacQueen. Some methods for classification and analysis of multivariate observations. In L. LeCam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, Berkeley, 1967. 97

[8] G. W. Milligan. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50:159–179, 1985. 97

[9] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a dataset via the Gap statistic. Technical Report 208, Department of Statistics, Stanford University, 2000. `http://citeseer.nj.nec.com/tibshirani00estimating.html`. 97

[10] C. van Rijsbergen. *Information Retrieval, 2nd edition*. Butterworths, London, UK, 1979. 99

[11] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parkankangas. SOM Toolbox for Matlab 5. Technical Report A57, Neural Networks Research Centre, Helsinki University of Technology, Espoo, Finland, April 2000. 97

[12] E. Vorhees. *The effectiveness and efficiency of agglomerative hierarchical clustering in document retrieval*. PhD thesis, Department of Computer Science, Cornell University, UK, 1985. 97

Self-Organising in Multi-agent Coordination and Control Using Stigmergy

Hadeli[1], Paul Valckenaers[1], Constantin Zamfirescu[1], Hendrik Van Brussel[1], Bart Saint Germain[1], Tom Hoelvoet[2], and Elke Steegmans[2]

[1] PMA Division, Katholieke Universiteit Leuven, Celestijnenlaan 300B, B-3001 Heverlee-Leuven, Belgium
{Hadeli.Karuna,Paul.Valckenaers,Constantin.Zamfirescu, Hendrik.VanBrussel,Bart.SaintGermain}mech.kuleuven.ac.be
http://www.mech.kuleuven.ac.be/pma/default_en.phtml
[2] DistriNet, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001, Heverlee-Leuven, Belgium
{Tom.Hoelvoet,Elke.Steegmans}@cs.kuleuven.ac.be

Abstract. In order to cope with today's dynamic environment, the described manufacturing control system is designed as a self-organising multi-agent system. The design of this novel system implements the PROSA reference architecture [1]. Coordination among agents is done indirectly through a pheromone-based dissipative field as is done by social insects in coordinating their behaviour. In this case, our agents act as social insects interpreting the pheromones put by the others in the environment. This control system is built from the basic elements of any manufacturing controller, namely products, resources and orders. However, the overall control system is constructed not only from those basic elements but also employing the appropriate interaction patterns among the agents who represent them. For coordination purposes, the agents send out a kind of mobile agents - artificial ants - to lay down information on the environment. In our case, where fulfilling the manufacturing orders is the main concern, there are at least 3 types of ant in this system: (1) feasibility ants - to propagate information concerning the feasible finishing routes; (2) exploring ants - to explore the feasible routes; and (3) intention ants - to propagate the route preferences. The overall mechanism enables the system to exhibit a self-organising behaviour.

1 Introduction

This paper presents research on self-organizing multi-agent systems for manufacturing control. Manufacturing control systems have to manage the internal logistics of their underlying production system; they decide about the routing of product instances, the assignment of workers, raw material, components, and the starting of operations on semi-finished products. Today, manufacturing control systems have to cope with an extremely dynamic environment — machine breakdowns, rush orders, late deliveries, new products, equipment upgrades, plant layout modifications, etc. As a consequence, it is desirable and even necessary to engineer and design them as self-organising systems.

© Springer-Verlag Berlin Heidelberg 2004

This paper discusses the self-organising capabilities in multi-agent manufacturing control research prototypes and their common underlying generic design. The research developments exploit a specific advantage that they have over many other multi-agent systems. A manufacturing control system exists relative to its underlying production system. Such control system continuously rediscovers this underlying system and adapts its own structure accordingly without centralised guidance. Because this underlying system is embedded in the real world, a society of agents reflecting parts of this underlying system only has superficial integration problems. To achieve this, the manufacturing control system prototypes employ the PROSA [1] reference architecture in which the main agents correspond to items (i.e. products, orders, resources) that exist in the world of their underlying production system (and not to functions).

The coordination mechanisms among the agents are inspired by the behaviour of social insects, more specifically by food foraging ant colonies. These mechanisms emergently provide information about remote and global system properties in local information spaces, typically attached to a resource agent. These mechanisms enable emergent, self-organising and system-wide coordination.

This paper first discusses self-organisation and emergent behaviour. Next, the multi-agent manufacturing control system design is presented. This section first discusses food foraging behaviour in ant colonies, and continues with a presentation of the structure of the control system, which has specific features supporting the implementation of coordination mechanisms based on insights on how the ant colony's technique works. The subsequent section presents the coordination mechanisms (inspired by the ant colony's behaviour) that are used for manufacturing control. Next, the self-organising properties of these mechanisms are discussed. Finally, two prototype implementations — which served to validate the coordination mechanisms — are presented, followed by conclusions and future work.

2 Self-Organisation and Emergent Behaviour

The essence of self-organisation is that a system acquires a spatial, temporal or functional structure without specific interference from the outside [2]. By “specific” is meant that the structure or function is not impressed on the system, but that the system is acted upon from the outside in a non-specific fashion. The organisation can evolve either in time or space, can maintain a stable form or can show transient phenomena. General resource flows into or out of the system are permitted, but are not critical to the concept. The research, discussed in this paper, conforms to the above definition in that the underlying production system only interferes by enabling reflection of itself in the multi-agent manufacturing control system (hence, only self-knowledge is required from the components of the underlying system).

A self-organising application basically is an application running without central control in which the different behaviours of the individual entities lead to an emergent coherent result [3]. Usually, such applications or systems take inspiration from biology, the physical world, chemistry, or social systems. Characteristics of these applications are their ability to accomplish complex collective tasks with simple individual behaviours, without central control or hierarchical structure.

The self-organising applications in this paper keep the individual behaviours simple in the sense that they limit the exposure of the individual entities to the overall system properties. However, the generic design for the manufacturing control systems allows these entities to become experts and to be very intelligent within their own scope. For instance, a resource agent can be an expert on the corresponding resource, but only knows the identity of the entities to which it is connected and of those that are residing at the resource. Note that requirements for the products on the machine (weight, size…) are part of the resource agent's self-knowledge. Consequently, the agents may need to be "educated" in the sense that they know basic physics and mathematics. The research applications also comply with the above definition in that the emergent behaviour leads to system-wide coordination well beyond the limited scopes of the individual agents.

Emergence is generally understood to be a process that leads to the appearance of structure not directly described by the defining constraints and instantaneous forces that control a system [4]. It denotes the principle that the global properties defining higher order systems or "wholes" (boundaries, organization, control…) can in general not be reduced to the properties of the lower order subsystems or "parts" [5]. It is the arising of novel and coherent structures, patterns, and properties during the process of self-organization in complex systems [6]. Moreover, emergent behaviour is a behaviour produced by the system that is more complex than the behaviour of the individual components in the system. This behaviour usually emerges from the interactions within the overall system.

The developments in this paper apply a number of insights in emergent systems. Firstly, must be a sufficiently large number of agents to be effective. The manufacturing control system creates agents that are virtual representatives of the physical entities in the underlying system at a selectable frequency. Thus, the number of entities in the physical world does not limit the number of agents responsible for the effective emergent behaviour in the system. Secondly, the design has self-reinforcing mechanisms. For instance, the propagation of intentions through the system coerces other agents to account for them, which is likely to enhance the optimality of the intended action plan. Thirdly, the agents constantly forget and refresh the information, which enables the emerging organisation to account for the dynamics in the system. Furthermore, the design complies with the four theoretical requirements to exhibit self-organising behaviour, which is formulated by the Nobel Prize winner Ilya Prigogine and his colleagues [7]; this is discussed toward the end of this paper.

3 Multi-agent Manufacturing Control

This paragraph describes the multi-agent control system structure. Conceptually, manufacturing systems can be divided into two parts, the physical resources and the control system. Physical resources perform the operations in manufacturing systems whereas the control system is managing the internal logistics. The control system decides about the routing of product instances, the assignment of workers, raw material, components, and the starting of the processes on semi-finished products. This manufacturing control does not control the manufacturing processes themselves (process control is seen to be part of the resource), but has to cope with the consequences of the processing results (e.g. the routing of products to a repair station). The research discussed in this paper investigates the development of multi-agent manufacturing control systems exhibiting effective self-organisation capabilities.

According to Parunak [8], when identifying agents for manufacturing system, agents should be things rather than functions. Moreover, when designing the agents, agents are designed in object-oriented way. In this approach, Jackson System Development methodology is used [9] to guide the design of an agent. When applying this methodology, the design is try to reflect the entities of interest, and furthermore, the functionality needed to answer the user requirements is implemented on top of this reflection of the world of interest. In this research, the manufacturing control systems implement the PROSA reference architecture [1]. According to PROSA architecture, there are three basic agents that needed to build a manufacturing control system: order agents, product agents and resource agents. Each of them is responsible for the logistic tasks, the process plans and the resources respectively.

These basic agents are structured using object-oriented concepts like aggregation and specialization. A further discussion of PROSA is given below. The most important property is that the main agents correspond to something that exists in the manufacturing setting; the basic PROSA agents are not functions.

The remainder of this paragraph first presents the source of inspiration for the emergent coordination techniques, food foraging in ant colonies, which explains why the certain features are made available in the control system infrastructure. Next, the main development steps to design and implement a specific control system are described. Paragraph 4 then presents the translation of the ant colony coordination mechanism into the manufacturing control setting.

3.1 Stigmergy and Coordination

In general, there are two basic ways to perform coordination. There are: coordination by direct communication, and communication within dissipative fields (indirect interaction). This research employs communication within dissipative fields. It utilizes an approach inspired by the way in which ant colonies propagate information while foraging for food; this is called stigmergy by biologists [10]. Stigmergy describes the use of asynchronous interaction and information exchange between agents mediated by an "active" environment. Investigations of social insect societies show that they coordinate themselves by producing such a dissipative field in their environment.

The interaction of ants is based on the existence of a smelling chemical substance called a pheromone. Ants deposit pheromone in their environment, which is observed by other ants and influences their behaviour. The pheromone depositing mechanism supports two major operations mediated by the environment in which the insects are situated, namely aggregation (accumulation of information) and evaporation (disappearing of old, possibly outdated information).

The coordination mechanisms in this paper are based on the food foraging behaviour in ant colonies. While foraging for food, ants follow a simple agenda, their behaviour being influenced by a permanently changing environment. Ants forage for food in the following way [11]:

- In the absence of any signs in the environment, ants perform a randomised search for food.
- When an ant discovers a food source, it drops a chemical smelling substance on its way back to the nest and creates with this activity a pheromone trail between nest and food source. This pheromone trail will evaporate if no other ant deposes fresh pheromones.
- When an ant senses signs in form of a pheromone trail it will be urged by its instinct to follow this trail to the food source, although there remains a small probability that the ant doesn't follow the trail. When the ant reaches the food source, it reinforces/refreshes the pheromone on its way back to the nest.

Attractive properties of this ant mechanism are that (1) the evaporation makes the colony to forget the information that is no longer valid, and (2) the environment is reused in the solution (no maps in the brain of the ants). Also note that the presence of a pheromone trail increases the probability that ants will find the food at its end, and reinforce the trail on their way back to the nest. Such self-reinforcing is seen to be a necessary property for effective emergent behaviour. Fig 1 illustrates how an ant operates during food foraging.

An important consequence for the control system infrastructure is that it must be possible to deposit, observe and modify information in relevant locations, where the lifespan of the information can be restricted. This is addressed, among other issues, in the next section.

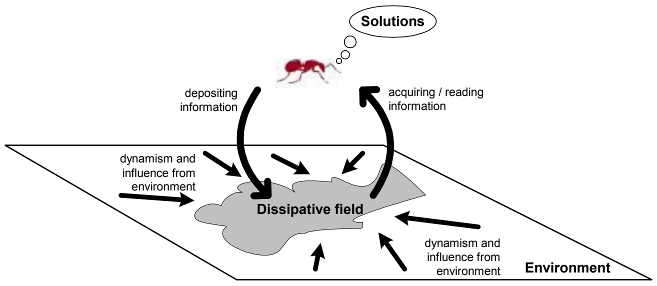

Solutions
depositing information
acquiring / reading information
dynamism and influence from environment
Dissipative field
dynamism and influence from environment
Environment

Fig. 1. How an ant behaves

3.2 Control system Infrastructure

In contrast to the ants, computing agents live in a world that is disjoint from the manufacturing ironware. An agent cannot observe nor impact upon the manufacturing system directly. Therefore, to implement manufacturing coordination using stigmergy, the manufacturing ironware has to be connected with the cyber world in which the agents reside. As a consequence, the first control system development step is to mirror the manufacturing system in the agents' world.

For every resource in the factory, an agent is created. This resource agent knows only about its corresponding resource and keeps itself in sync with reality. Moreover, this agent only has local expertise and uses the structure of the agent system to distribute global information (see further). The same applies to every order arriving in the system. Likewise, product types are reflected in product agents. These agents reside in the control system. Note that these agents reflect their connections and location. Resource agents know which of their exits is connected to which entry of which other resource, and vice versa. Order agents know on which resources the corresponding product pieces are located and vice versa. This allows computing agents to navigate through this reflection of the underlying production system in the cyber world. Moreover, these reflecting agents provide information spaces on which other agents can deposit, observe and modify information (pheromones), where the lifespan of this information is finite (evaporation).

Through this reflection, the architecture supports three concepts for the implementation of ant colony coordination [12]:

- Artificial ant agents as the acting individuals (see further)
- Artificial pheromones as the information carriers (see below)
- A distributed pheromone propagation infrastructure to create the artificial dissipation field.

The above creates a suitable environment providing mechanisms for the ant agents to navigate through the control system and to maintain pheromone information within the control system. The environment provides following architectural concepts [13]:

1. “Locations”, where agents can reside and retrieve or contribute information.
2. A management mechanism for storing and maintaining artificial pheromone objects at these locations.
3. A propagation mechanism to allow the agents to move within such a topology of locations for the purpose of deposing or sensing information.

Finally, as mentioned earlier, such a manufacturing control system is realized as an implementation of the PROSA reference architecture. The PROSA reference architecture describes the essential manufacturing entities and their roles in manufacturing domain [1]:

- A resource agent reflects a physical part, namely a production resource of the manufacturing system, and an information processing part that controls the resource. It offers production capacity and functionality to the surrounding agents. It holds the methods to control these production resources to drive production. A resource agent is an abstraction of the production means such as a factory, a shop, machines, conveyors, pallets, components, raw materials, tools, personnel, etc. On the other hand, to facilitate the implementation of the stigmergic approach, resource agent also provides a kind of *blackboard* or location where information can reside and is accessible for interested agents.
- A product agent holds the process and product knowledge to assure the correct making of its product with sufficient quality. A product agent contains consistent and up-to-date information on the product life cycle, user requirements, design, process plans, bill of materials, quality assurance procedures, etc. As such, it contains the “product model” of the product type, not the “product state model” of one physical product instance being produced. The product agent acts as an information server to the other agents in the system.

- An order agent represents a task in manufacturing system. It is responsible for performing the assigned work correctly and on time. It manages the physical product being produced, the product state model, and all logistical information processing related to the job. An order agent may represent customer orders, make-to-stock orders, prototype making orders, orders to maintain and repair resources, etc. Often, the order agent can be regarded as the work piece with a certain control behaviour to manage it to go through the factory, e.g. to negotiate with other parts and resources to get produced.

Summarizing, the development of a PROSA multi-agent control system starts with the implementation of a society of agents that reflect entities in the underlying manufacturing system (tasks, product types and resources), where these agents are equipped with information spaces (especially the resource agents). The resource agents support virtual navigation by providing a model of their local connectivity. In this virtual world within the control system, to be kept in sync with the real world, the agents create ant agents that travel and navigate across this cyber world, collecting information and depositing it on the information spaces. Through this activity, the ant agents create dissipative fields that mutually influence their behaviour and makes patterns emerge. How the basic PROSA agent creates ant agents and how these ant agent perform their task in an emergent and self-organising manner is discussed in the next section.

4 Self-organising Coordination

This paragraph presents the coordination mechanisms inspired by the food foraging in ant colonies. In the research prototypes, there are three information-propagating coordination activities (in future systems, it is likely that there will be more). These coordination activities, called control task, are the:

- Feasibility information propagation — information at the locations throughout the system enables the order agents and their ant agents to avoid routes that fail to guarantee that their process plan can be properly executed.
- Exploring task — order agents create ant agents that search for attractive routes to get themselves produced, at a suitable frequency.
- Intention propagation task — order agents create ant agent that virtually travel through the factory and inform the appropriate locations about the orders' intentions, again at a suitable frequency to account for changes.

Based on the information from these three coordination mechanisms, order agents make the choices that determine what will actually happen when the moment of commitment is sufficiently close. The actual control actions result from these choices by the order agents and the execution of these choices by the resource agents.

So-called ant agents perform the propagation of information in these coordination layers. The basic PROSA agents create ant agents in the prototypes at a suitable frequency. For instance, an order agent may create an intention-propagating ant agent every ten seconds. Ant agents navigate virtually through the reflection of the underlying production system in cyber space; they move faster than the real system by several orders of magnitude. In contrast, the basic agents accompany their real-world counterpart, evolve much slower and often need a transactional implementation when controlling an actual production system.

4.1 Feasibility Information Propagation

Starting from the resource agents that correspond to factory exits, a type of mobile agents called *feasibility ants* are created at a given frequency; this frequency is called the refresh rate. A feasibility ant exhibits the following behaviour:

- Firstly, the ant acquires from the corresponding resource agent a description of the processing capabilities of the resource on which it virtually resides. At the initial resource, this typically would be shipping operations.
- Secondly, this description is merged with the information on processing capabilities that has been collected so far. Initially, this will be "no processing capabilities".
- Thirdly, the ant agent obtains the list of entries to its current location. The ant creates sufficient clones of itself to ensure that there will be one ant agent per entry. One ant now navigates in the virtual counterpart of the underlying production system to the corresponding exit of each of the preceding resources. Note that resources corresponding to factory entrances have zero entries.
- Fourthly, the ant observes on the information space attached to the exit which processing capabilities information was deposited by its own generation of ant already, and merges this with its own information. Indeed, ants arriving through different routes may visit the same exit. If the merging of the information results in the information already available, the ant dies. Otherwise, the ant deposits the new information, overwriting the old one, moves to the resource of the exit and starts the first step of its behaviour pattern.

Additional measures are taken to ensure that the number of ant agents is restricted (cloning budget) and that the ant's journey is finite (hop limit), even when programming mistakes are made in the merging of the feasibility information. The feasibility information remains valid for a small number of refresh cycles only (e.g. three cycles).

When order agents or their ant agents navigate forward (downstream) through the factory, these agents retrieve the feasibility information attached to the exits under consideration. They forward this information to their product agent who will reply whether the exit is eligible from a feasibility point of view (all required operations will eventually be found in a suitable sequence without any guarantees on optimality). Thus the feasibility layer restricts the search space to routings that are technically feasible.

Thus, the feasibility layer is composed of agents with a limited scope where the emergent behaviour makes it possible to account for system-wide feasibility constraints. The solution makes the environment's reflection part of the solution. The refresh-and-forget mechanism handles the changes in the underlying system.

4.2 Exploring Task

In this paragraph, the discussion assumes for simplicity reasons that order agents can be associated with a single work piece moving through the factory. However, the exploring mechanism can be and has been applied to more complex situations in which order agents correspond to multiple work pieces (shopping list), work pieces that are cut in multiple pieces, multiple pieces that are assembled… Details of such implementations are outside the scope of this paper.

Order agents are responsible to get their product instance produced in time. An order agent has to route its product instance through the factory such that it may undergo all the required processing steps. To explore the available possibilities, order agents create mobile agents called *exploring ant agent* that virtually navigate through the factory starting from the current position of the work piece until the end. Exploring ant agents are created at a suitable frequency, and each ant agent investigates the expected performance of a single feasible route of the product instance through the factory; exploring ant agents do not clone themselves.

While navigating virtually through the factory and having themselves virtually produced, the exploring ant agents query the resource agents, which they encounter on the selected route, about the expected performance. For instance, they inform a transport device agent about the time they expect to be available for transport at one location and provide a desired arrival time at another location. The transport device agent replies with the expected arrival time at the target location. This can be later than desired if the transport device expects to be congested at that time. Likewise, the exploring ant agent gets expected times for the operations it requests at the processing stations on its route. Notice how little the exploring ant agents need to know about the factory on which they reside (simple components but emergent self-organising overall behaviour).

When arriving at the end, the exploring ant agent reports back to the order agent which route it explored and the order agent computes the expected performance of this route. The order agent maintains a selection of well-performing routes that have been recently explored (evaporation). The mechanisms that decide which routes will be explored and how frequently a route will be explored (refresh) are outside the scope of this paper. Note however that there is a very active research domain addressing this aspect [11].

4.3 Intention Propagation Task

When the order agent has collected sufficient information about the possible routes and their expected performance, the agent selects one ‘best-performing' route. This route becomes the order agent's intention (strategy) to have all the operations accomplished.

At regular intervals, the order agent creates another type of mobile agent called *intention ant agent* that virtually navigates through the factory along the intended route and informs the resource agents along this route about the order agent's intention. An intention ant agent differs from an exploring ant agent in two ways. First, it has a fixed route to follow. Second, it makes the appropriate bookings with the resources. Indeed, when resource agents are informed about the intentions of all orders that intend to visit them, they are able to construct a local work schedule (local load forecast) for themselves covering the near future. This local schedule is used to give accurate answers to the exploring ant agent and the intention ant agent alike. Note that resource agents may have local authority on how well they serve requests from the visiting order agents and their ant agents (priority rules, change-over avoidance…) and they often do.

The evaporate-refresh mechanism is also applied to the propagation of intentions. When an order agent fails to refresh, the corresponding booking will be removed from the local work schedule of the resources involved it. Moreover, the refresh serves to keep the order agent informed of the expected performance of its currently intended route. When arriving at the end, the intention ant agent reports back the performance of the current intention of the order agent. If the situation in the factory changes, this performance can change dramatically (e.g. due to machine breakdown on the intended route or the arrival of rush orders taking priority) in the positive and negative sense.

The exploring behaviour continues in parallel with the intention refreshing. As a consequence, the exploring ants may report back more attractive routes than the current intention. When this happens, the order agent may change its intention. However, it is important to impose socially acceptable behaviour on the order agents in this respect: once intentions have been made known to the relevant parts of the overall system, these intentions are not changed lightly. In other words, the perceived amelioration must be significant before intentions change, the frequency at which intentions change should be restrained, and the occurrence of such a change should be stochastically spread across sufficient refresh cycles.

The latter ensures that when a large number of orders are affected by a disturbance, only a small percentage of them will react to the disturbance whereas the other agents see the effect of this reaction before changing their own intentions. For instance, when a machine breaks down, the shift to alternative routes avoiding this machine does not result in a stampede overloading these alternatives. Instead, the affected orders virtually trickle toward the alternatives until the heavy loading of these alternatives balances with the estimated repair time of the broken machine.

4.4 Execution

The exploring and intention-propagating behaviour of the order agents occurs at a much higher speed than physical production. In other words, the control system virtually produces the required production thousands of times before the actual production activity is triggered. How much computational effort is spent depends on the economic value of better routing and sequencing decisions in a given factory; this differs from factory to factory and depends on the capital investments and the nature of the manufacturing processes (i.e. how sensitive these processes are to proper scheduling and how difficult are the scheduling tasks).

In any case, the order agents monitor the progress of the corresponding physical product instance and when the time approaches, the order agent executes the next step in its current intention. Deviation from the current intention can be seen as changing this intention at the last moment. Socially acceptable behaviour implies that this only happens for very serious reasons. Indeed, changing of intentions is more acceptable for decisions that are far in the future when other agents still have the opportunity to adapt and react.

4.5 Remarks

The above discussion describes only a minimal implementation. In practice, there may be additional layers. An example is a layer that indicates batch-building opportunities in which orders make their presence and batch compatibility known through some similar information propagation mechanisms supporting evaporate-refresh. Likewise, orders that need to arrive just in time have to reverse the direction in which routes are explored.

Overall, the interplay of PROSA agents and ant agents enables the control system to impose feasibility constraints and to provide short-time predictions of how the manufacturing system will perform without imposing a system-wide organisation. The control system adapts automatically to the underlying production system and accounts emergently for changes and disturbances. Each agent has a limited scope yet contributes to a system-wide coordinating behaviour. It suffices to make each agent reflect his aspect/scope correctly and adequately. The next section discusses how this multi-agent system design complies with more universal insights in emergent systems.

5 Emergence and Self-Organisation

To exhibit a self-organising behaviour, a system has to fulfil at least four theoretical requirements mathematically derived by the winner of a Nobel Prize for chemistry Ilya Prigogine and his colleagues [7], namely:

1. At least two of the components in the system must be *mutually causal*. A system exhibits mutual causality if at least two of the components in the system have a circular relationship, each influencing each other.
2. At least one of the components in the system must exhibit *autocatalysis*. A system exhibits autocatalysis if at least one of the components is casual influenced by another component, resulting in its own increase.
3. The system must operate in a *far-from equilibrium condition*. A system is defined as being far-from equilibrium condition when it imports a large amount of energy from outside the system, uses the energy to help renew its own structures (autopoeisis), and dissipates rather than accumulates, the accruing disorder (entropy) back into the environment.
4. To exhibit *morphogenetic changes*, at least one of the components of the system must be open to external random variations from outside the system.

The discussion below uses these definitions as a guidance to discuss the multi-agent manufacturing control technology presented earlier.

Firstly, the order agents, together with the ant agents created with them, are mutually causal. Indeed, the intentions propagated by one order agent affect the intention selection of other order agents and vice versa. This effect is even more pronounced if the manufacturing control system is further elaborated in industrial case studies in which processing equipment needs set-ups and changeover when the sequencing of products fails to fits the current configuration. A booking by the intention propagating ant agent will cause the resource agent to give favourable performance Fig.s to exploring ant agents that correspond to a product that can be processed in the same set-up whereas orders for dissimilar products receive unfavourable responses.

Secondly, the above example also illustrates that autocatalysis is quite common in the manufacturing control design. The presence of bookings through intention-propagation makes the other orders take this into account, which in turn is likely to reinforce the optimality of these intentions. Indeed, the reserved time slots on the resource remain available whereas the neighbouring slots become occupied. Similarly, a layer that indicates batch-building opportunities will cause compatible orders to select routings that bring them together in front of the batch processing equipment, which in turn makes the indication of batch building around such emergent cluster stronger as well as weaker elsewhere, attracting even more batch members.

Thirdly, a manufacturing control system is an extremely open system in which materials, energy and information continuously flow in and out. The control system brings structure in this flow through the autocatalysis described above, and continuously adapts its behaviour/state/structure in response to these flows. For instance, the collection of order agent constantly changes in function of the external elements. Importantly, the systematic application of an evaporate-refresh mechanism draws significantly on computing powers and contributes to the ability of the control system to adapt itself to changed situations.

Fourthly, there is no lack of random variations with equipment malfunctioning, variable processing results and durations, and market forces causing hard-to-predict production demands. Also, when the underlying system is modified, the structure of the reflection in resource agent changes accordingly.

Furthermore, other insights exist. For instance, the mechanisms used by social insect require a sufficiently large number of agents to be effective. The virtual nature of the ant agents solves this in the manufacturing control systems.

6 Prototypes

In practice, it is impossible to implement novel manufacturing control system designs directly in real production plants. It is prohibitively expensive, very risky, and it is impossible to repeat experiments when trying to compare alternatives. Therefore, the real plant is replaced by an emulated system, asynchronously connected to the manufacturing control system. The emulation mirrors the behaviour of the underlying production system such that the control system cannot distinguish it from being connected to a real factory (see Fig. 2.). As a consequence, replacing the emulation with a real factory requires negligible adaptation efforts for the control system software (note that connecting a physical resource to a computer network often requires significant effort).

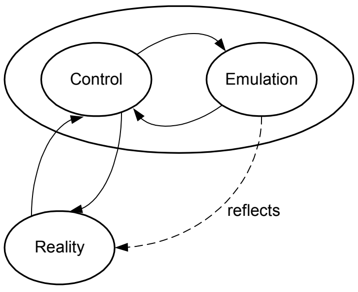


Fig. 2. Relation between control-emulation and reality

In this setting of an emulated production system offering connections over a computer network as if it were a real production system, multi-agent manufacturing control system prototypes have been and are developed.

Several prototypes have been built to validate the novel concept. The prototypes range from simple ones to real industrial case studies. One simple prototype controls a simple manufacturing plant that consists of an arrival station, an exit station and 11 processing resources connected [13]. In this prototype, several orders are sent to the system. Each order has a different operations sequence to follow and some operations have alternative resources to be processed. During the run time, one of the machines is randomly selected to be down, and the operation time for each operation is following certain distribution. The result shows that intention-based forecasting works in this case; moreover, the system is flexible enough to react to disturbances and to adapt to changes in the system environment. For instance, in the experiment on this system, there are 5 orders entering the system. For example, take a focus on the 4th order. The number 4 order has an operation sequence as follows: {Op1, Op3, Op5, Op6}. Early intention shows that those operations will be processed in R2, R4, R6 and R12 respectively. Nevertheless, order 4 does two intention changes. At time t_1=171.49, the 4h order discovers that the real step 1 (Op1) processing time is longer than the estimation. Consequently, re-calculation is done to update the possible start time for the remaining steps. Hence, on the next exploration, the ant discovers that the finishing time of its 3rd step at R6 becomes less interesting (t_{f6}=536.242), and this triggers the willingness to switch its intention to R9, which offers a better solution (t_{f9}=533.475). The second intention change happens at time t_2=277.493. This is due to the shorter real processing time for the 4th order's step 2 (Op3), and consequently the remaining operations can start earlier. Therefore, on the next exploration the ant finds out that R6 is the resource that offers the best solution for its next operation. This condition triggers the ant's intention to switch to R6 (finishing time in R6 is t_{f6}=521.005, in contrast, finishing time in R9 is t_{f9}=531.005). See Fig. 3.

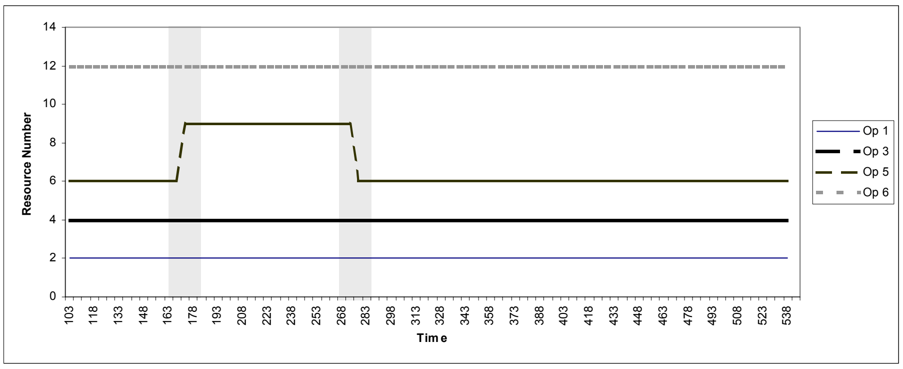


Fig. 3. Intention changes chart

The other prototype is addresses a real industrial case. This prototype reflects the real conditions of a department in a factory that produces weaving machines – some data on the case has been modified for confidentiality reasons. This department is an open job-shop plant for weaving machines components. The architecture of the shop floor is shown in Fig. 4. This shop is composed of 13 processing resources and 1 exit station. For storage of finished or work-in-process products, an automatic storage and retrieval system (ASRS) is installed in the middle of the shop and connected with rail-based transport device, called the tram. All resources are connected with the tram, where its task is to transport the required parts from workstation to workstation/storage and vice versa. In this prototype, a study on the architecture, coordination behaviour and decision making process was done in a more intensive way. This prototype addresses several issues, namely how an agent-based manufacturing controller can preserve its functionality against highly customisable modular plant architecture, the ability to cope with disturbances in production resources, factory organization and planning processes [14]. A set of experiments was carried out on this prototype. Numbers of orders are released to the shop floor. Consider order R00. To produce R00, it has to undergo a number of operations; one of them is the FINISH operation, that is the last operation. Along its lifecycle, order R00 keeps on sending mobile ants to explore the network and express his intention. The accuracy of the starting time forecast for operation FINISH of order R00 is shown in Fig 5. It shows that the forecasting is converging as its approaches the real execution time. The sudden increase in the deviation between the estimate and the actual execution reflects the occurrence of a disturbance (e.g. a machine breakdown or a rush order arrival). The inset graph shows the result of exploration in detail for the last 250 time units.

On the other hand, inside every order agent that has released ant agents to explore the shop floor, already has the strategy on how to get all the production steps executed (see Fig. 6). This strategy is not the definitive solution, because as the shop floor changes from time to time, the solution also has the possibility to alter. The black mark in the graph shows concluded operation, on the other hand is the future operations to be finished.

Furthermore, since an intention ant propagates intention by reserving time slots in resource agents, inside one resource agent, there is a load forecast that can be represented as Fig. 7. This graph shows the load forecast at certain resource (workstation). The load represents the orders' intention. It shows when the orders will be processed in this resource and the duration.

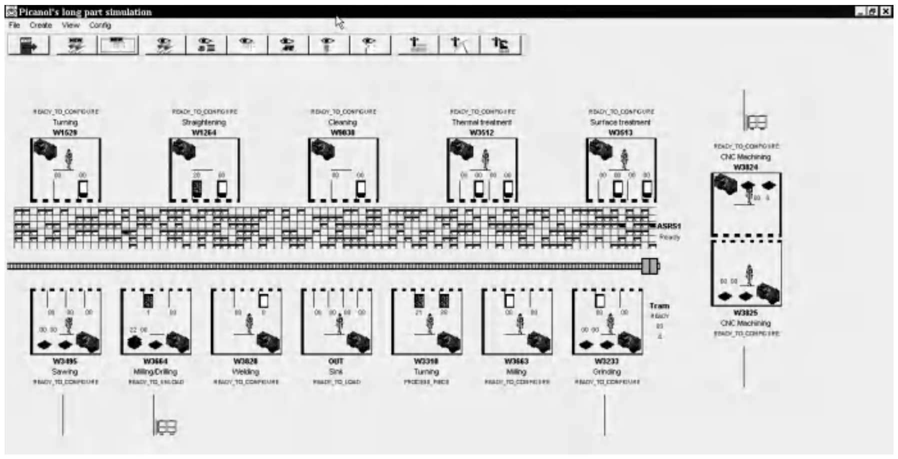


Fig. 4. Real industrial case prototype

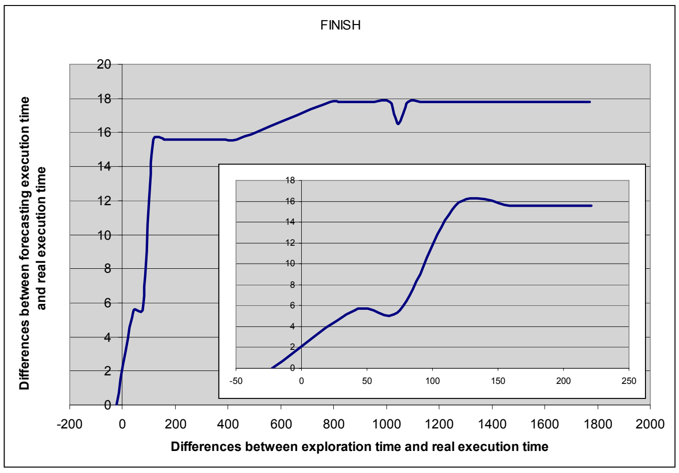


Fig. 5. Accuracy of the emergence solution for FINISH operation of order R00

Both prototypes employ the same underlying principles described in the previous chapter. They proved to exhibit self-organizing capabilities in order to accommodate the requirements of the incoming orders in compliance with the occurring disturbances. In the first prototype the disturbances are captured in the emulation model, while the second one offers the possibility to introduce through the user interface any disorder that is not restricted by the physical reality. The solution to achieve the plant objectives emerges from the indirect interaction among the agents through the pheromone environment, which is dissipated afterwards by the employed refreshing algorithm.

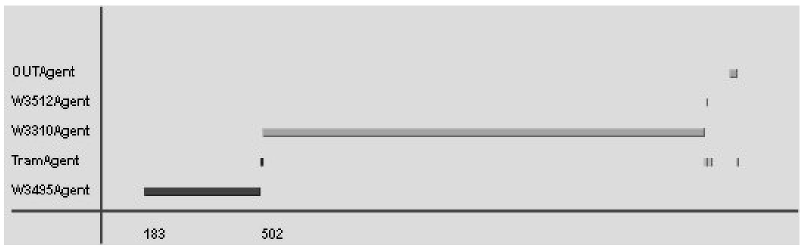


Fig. 6. The way to finish strategy of certain order

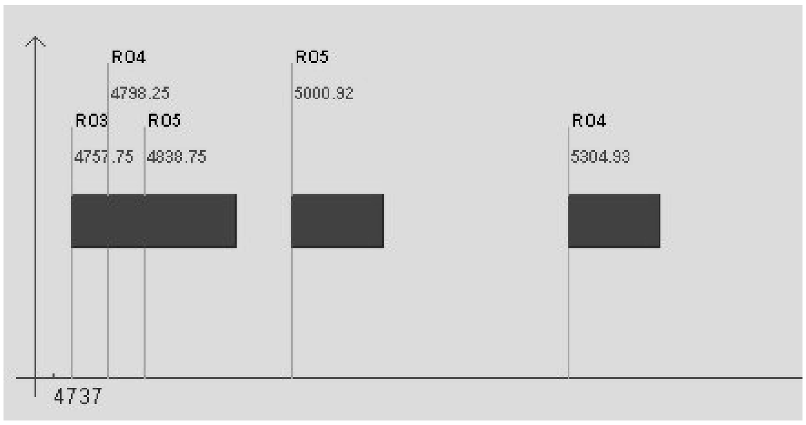


Fig. 7. Emergence load forecasting in resource

7 Discussion and Conclusion

This paper presents a novel manufacturing control design employing the mainstream development approach from object-oriented technology: entities/agents in the design reflect real-world entities and not the system functionality. This modelling approach guarantees that the components of the system are not over-exposed to the complexity of the overall system. It also provides reusability, separation of concerns, robustness and flexibility.

Pheromones in this system play important role in this system; they help order agents in accomplishing their task, and enable agents in this system to react without performing direct communication. Pheromones in this system can be found in the form of information that is deposited by feasibility ants at the exits of each resource agent. In addition, load forecast, production capacity and functionality information is placed in resource agent's blackboard, being the other type of pheromone information in this system.

The other interesting aspect is that the design of this novel manufacturing control reveals self-organising behaviour. According to Heylighen, there are at least 7 characteristics of self-organising systems, namely [15]:

- *Global behaviour from local interactions*. This system is not only the unification of product, resource and order agents. The coordination behaviour between the agents builds a bigger system than the sum of all elements.
- *Distributed control*. Control in this multi-agent system is distributed and not centralized. Every entity in this system has its own responsibility. The limitation of the knowledge owned by each entity makes it unable to the mind the other agents' businesses.
- *Robustness and resilience*. The design of the construction guarantees that removal of some parts will not seriously damage the whole system. Whenever needed, new elements such as a new resource agent that represents a new resource in the shop floor can be plugged in and plugged out. Any changes in the structure will be immediately notified; hence the control system can accustom the changes and provide more flexibility to the system.
- *Non-linearity and feedback*. Small changes in this system, such as machine breakdown, changes in processing time, can cause the system to react differently. Those changes will affect on any orders in the system. Moreover, these changes also trigger the refreshment of the existing control layers. Concerning about feedback, the content of control layers is different from time instant to time instant. For instance, any decision-making performed by an order agent (either when propagating intention or arriving at resource(s)) will provide a feedback to the state of the resource's schedule. This feedback will affect the results from next refresh cycle of the exploring and intention layers.
- *Organizational closure, hierarchy and emergence*. Coordination between the three types of agents and the propagation of information using ants has created a new individual and coherent whole. Interaction between ants through depositing and interpreting pheromones ensures that the entire agents in the system are well organized. This coherent whole system itself has its own behaviour that different from the behaviour of every single agent or ant. For instant, the coherent whole system will define how a new order that arrives will be treated. The treatment of this new order is not depending on single resource, but it has to account the existence of other orders that have arrived before and also the state of the system by that time. The properties and behaviour of this whole system is emergent.
- *Bifurcations and symmetry breaking.* Disturbances that happen in manufacturing systems can lead order agents to over-react through a continuous changing of their intention. When every order agent in the system changes its intention every time it figures out a more attractive solution, the system may become too nervous and might become uncontrollable. Therefore, a kind of social behaviour has to be implemented in order to control the intention changing mechanism. The decision taking mechanism of the order agents reflects this characteristic. Each order agent has his own decision criteria. Initially, order agent treats any alternative solution equally; nevertheless the existence of other orders and variety of decision criteria owned by every orders force the order agent to prefer certain

solutions to others, and this will finally lead to choose only one preferable solution. This solution becomes the order agent's intention, which ant agents propagate through the system. Once this intention has been made known to other agents, the order agent will be reluctant to switch to other solutions and requires a significant enhancement of perceived performance to effectively switch. Evidently, the order agent's initial choice may depend on minimal difference between perceived performance of the available alternative; it may even be a random choice between equally attractive solutions. However, once a path has been selected and propagated, the current intention is no longer equal; it receives privileges. Furthermore, randomisations in the intention-changing and decision-making mechanisms explicitly create asymmetries even when order agents and their ants find themselves in identical situations.

- *Far-from-equilibrium dynamics.* Arrival of new orders to the system in unpredictable sequence and frequency and also the install and uninstall of new resource(s) guarantee that the system behaves in a dynamic manner and has a far-from-equilibrium behaviour.

The described manufacturing control system exhibits self-organisation behaviour that enable the system to cope with and properly react to the dynamism of the manufacturing environment (i.e. machine breakdowns, rush orders, new product type, etc.) in a robust way. Nevertheless, a series of tuning parameters, like the frequency of ant propagation, evaporation rate, and intention changes will be further investigated in order to achieve a better design of this control system.

Furthermore, result from the prototype also shows that this system is able to perform a load forecast for every resources in system; moreover, for every order that enters the system a strategy to finish can be forecasted. In addition, each order is able to react to disturbances and changes in system, and to adapt its strategy.

Acknowledgement

This work was supported by the European Commission under the GROWTH program and the K.U.Leuven research council (GOA–AgCo$_2$)

References

[1] Van Brussel, H., J. Wyns, P. Valckenaers, L. Bongaerts, P. Peeters, Reference architecture for holonic manufacturing systems: PROSA. Computers In Industry 37 (1998)

[2] Haken, Herman. Information and Self-Organization: A Macroscopic Approach to Complex Systems, Springer-Verlag, Germany (1988)

[3] ESOA WG – Mission Statement http://cui.unige.ch/~dimarzo/esoawg/mission.doc

[4] Crutchfield, James P., Is Anything Ever New? Considering Emergence, SFI Series in the Sciences of Complexity XIX, Addison-Wesley, (1994)

[5] Heylighen F., Self-organization, Emergence and the Architecture of Complexity, Proceedings of the 1st European Conference on System Science, Paris (1989)

[6] Goldstein, J., Emergence as a Construct: History and Issues, Emergence, Vol 1, Issue 1 (1999)
[7] Glansdorff, P. & Prigogine, I., Thermodynamic Theory of Structure, Stability and Fluctuations. Wiley, New York, 1978.
[8] Parunak, H. V. D., Sauter, J., Clark, S., Toward the Specification and Design of Industrial Synthetic Ecosystems, Fourth International Workshop on Agent Theories, Architectures and Languages (ATAL) (1997)
[9] Jackson, M., Software Requirements and Specifications. Addison-Wesley, Amsterdam (1995)
[10] Theraulaz, G., E. Bonabeau, A Brief History of Stigmergy, Artificial Life, 5 (1999) 97-116
[11] Dorigo, M., Di Caro, G., & Gambardella, L. M., Ant Algorithms for Discrete Optimization, Artificial Life 5 (1999) 137-172.
[12] Valckenaers, P., Kollingbaum, M., Van Brussel, H., Bochmann, O., Short-term forecasting based on intentions in multi-agent control, Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference, Tucson (2001)
[13] Hadeli, Valckenaers, P., Kollingbaum, M., Van Brussel, H., Multi-Agent Coordination and Control Using Stigmergy, accepted for publication in Journal Computers in Industry
[14] Zamfirescu, C., Valckenaers, P., Hadeli, Van Brussel, H., Saint-Germain, B., A Case Study for Modular Plant Architecture, Lecture Notes in Artificial Intelligence 2744 (2003) 268-279
[15] Heylighen, F., The Science of Self-Organization and Adaptivity, The Encyclopedia of Life Support Systems, 2002.

Managing Computer Networks Security through Self-Organization: A Complex System Perspective

Noria Foukia[1] and Salima Hassas[2]

[1] University of Geneva
rue Général Dufour 24, CH-1211 Geneva 4, Switzerland
[2] LIRIS, Nautibus, 8 Bd Niels Bohr, Université Claude Bernard-Lyon 1
43 Bd du 11 Novembre F-69622 Villeurbanne

Abstract. The present paper proposes a new perspective to deal with computer networks security. Networks can be viewed as complex systems that exhibit self-organization and self-control properties. These properties are well suited for today's open networks like the Internet. In such uncertain environment as the Internet, ensuring survivability is a hard task. A parallel is made with natural life systems that also have to survive external aggression and that also exhibit complex system characteristics. This paper describes our research work dealing with complex architecture for Intrusion Detection and Response System (IDRS). In the perspective of complex systems, the proposed IDRS presents self-organization characteristics based on interaction between single entities. These entities are implemented using Mobile Agents (MAs) that incarnate a complex "artificial ecosystem" to detect and to answer intrusions.

Keywords: Self-organization, emergent behavior, swarm intelligence, software engineering, complex networks, intrusion detection and response, mobile agents.

1 Introduction

Computer networks have gained distribution at a large scale, a strong dynamic connectivity and a high openness. These properties while being of a great benefit - since they guarantee facilities of a wide access to information, services and computing resources - make networks management and securing very hard to ensure. Recent research [5][3] states that "complex man-made networks such as Internet and the world wide web, share the same large-scale topology as many metabolic and protein networks of various organisms" and that "the emergence of these networks is driven by self-organizing processes that are governed by simple but generic laws". Topology of the Internet has been recently characterized as scale free [15]. A scale free network is a class of an heterogeneously wired network. The topology of these networks is extremely sparse with a few randomly distributed

© Springer-Verlag Berlin Heidelberg 2004

highly connected nodes (hubs) and many nodes with few connections. These networks also exhibit the so-called small world phenomenon [20], which means that they are characterized by a short average path length, implying that each node of the network could reach any other node through a relatively small number of hops. These characteristics make the Internet surprisingly robust face to random attacks and failures and in counterpart extremely vulnerable to strategic attacks directed to nodes with high connectivity [1][14]. Consequently, securing such networks requires highly flexible and adaptive mechanisms.

In [15], Albert and Barabasi present a substantial review of theoretical models and empirical studies conducted on the characterization of complex network. We describe in the following section some of their reported conclusions. We then motivate the need for a self-organizing approach for the management of security in such networks. We present our approach to address this issue and illustrate it through a case study in the context of distributed intrusion detection and response.

2 Topology of Complex Networks

In mathematical terms a network is represented by a graph G={N,E} where N is a set of nodes and E a set of edges. An edge is a link between two nodes. As described in [15], complex networks have been studied through different models with respect to three characteristics:

– degree distribution: which is the distribution function P(k) representing the probability that a randomly selected node has exactly k edges.
– clustering coefficient: which is a measure C_i associated to a node i, which represents the ratio between the number E_i of edges that actually exist between the set of nodes i and its k_i first neighbors, and the total number of possible edges between them if the node i and its k_i neighbors were completely connected to each other ($C_i = \frac{2E_i}{k_i(k_i-1)}$)
– connectedness and diameter: the diameter of a graph is the maximal distance between any pair of its nodes. Connectedness in a given graph is often measured by the average path length between any pair of nodes.

These characteristics give some indications about the graph topology and the underlying organizing process, leading to its topology. Traditionally large scale networks, where represented by random graphs through the Erdos-Renyi model. Recent work [20] showed that real complex networks do not behave like random graphs. In particular for a large number of networks like the WWW and Internet [5][3], the degree distribution follows a power law: $P(k) = k^{-\gamma}$, which expresses the coexistence of a few highly connected nodes with many other nodes with a low connectivity. As already mentioned, these large scale networks are called "scale free" and as complex networks they exhibit the so-called small world phenomenon [20].

For the rest of the paper, we will retain the following important points:

– small world networks are characterized with a short average path length and a high clustering coefficient. Concerning their degree distribution, they behave like random networks. Their topology is homogeneous with all nodes having approximately the same number of edges.
– scale free nature of complex networks is rooted in two main principles: *evolving growth*, induced by continuous adding of new nodes and *preferential attachment*, which expresses the fact that highly connected nodes are more likely to be connected to new arriving nodes, than other nodes.

These characteristics and properties are shared by many real existing networks: from metabolic and protein networks to social accointance networks [3]. As topology of such networks seems to be driven by underlying self-organizing processes [3], one has to endow them with mechanisms of control and management that exhibit this self-organizing characteristic.

3 A Self-Organizing Approach for Complex Network Security

3.1 Survivability and Safety in Computer Networks

Survivability refers to the capacity a system has to ensure its vital functionalities, when faced to attacks, accidents or perturbations that come through its environment. In the context of networks, this means that the network must be able to provide essential services during all its running. These services have to ensure a minimum working of the vital capacities of the network even under influence of any perturbation. For instance, if a successful intrusion or a fatal accident occurs. In an open system, the environment is uncertain, and is subject to perturbations that are unknown a-priori. Thus, to guarantee its survivability, an open system needs to be endowed with mechanisms such as those characterizing living organisms: self control, self-organization, self-adaptive behavior.
In addition, as mentioned in Section 1, the scale free characteristic of large scale networks renders the Internet vulnerable face to strategic attacks directed against vital nodes (hubs). We thus need to dispose of flexible mechanisms allowing the deployment of adaptive strategies for the network security.

3.2 Managing an Uncertain and Evolving Environment

In such an uncertain environment as the Internet, ensuring survivability is a hard task. The managing mechanisms must allow the follow up of the perpetual evolution of the environment. In the context of intrusion detection for example, new kinds of attacks are produced as frequently as old ones are treated. To be efficient, a mechanism of intrusion detection needs to be adaptive. It should exploit information on past attacks to discover new ones, and explore new directions to anticipate new attacks. This concept is what J. Holland called " balancing exploration and exploitation in adaptive processes" [10]. The problem is the effective use of limited resources, when exploring the research space of solutions to a given problem, under the constraints of evolving information. In this context, a balance between two strategies must be achieved.

– Exploiting promising possibilities in a self-catalytic way. The more a possibility is promising, the more it is exploited.
– Exploring new possibilities to diversify the search and discover new promising possibilities.

In nature, ants foraging behavior as well as immune system recognition mechanism, seem to maintain such a balance between exploitation and exploration as mentioned in [12]. During foraging, ants explore randomly their environment. When they find a source of food, they enforce its exploitation through spreading and scenting a chemical substance called pheromone. As mentioned in [12] "the immune system like ants colonies, combines randomness with highly directed behavior based on feedback. Immune system exploits the information it encounters in the form of antigens, to anticipate the recognition of potential new ones. But it always continues to explore additional possibilities that it might encounter by maintaining its huge repertoire of different cells".

In this state of spirit, Section 5 illustrates a system, we propose, for intrusion detection and response that uses balancing exploration and exploitation as a self-control strategy, in a context of computer security. We use the immune system inspiration for detection and the ants foraging behavior for response. This kind of strategy seems to be particularly adequate for information spreading through scale free networks. Indeed the exploration behavior permits to locate highly connected nodes (hubs), whereas the high connectivity of these nodes, allows the reinforcement of the exploitation strategy as we will describe it in the Section 4.

3.3 Our Proposal: An Artificial Ecosystem Inspired by Nature

We propose a framework based on the following main points:

– a collective behavior based approach, using the multi-agents paradigm where agents are situated on the network, and exhibit a behavior directed by a sensing/acting reactive loop. The whole system (network and agents) behaves as an artificial ecosystem.
– a self-organized behavior inspired by natural systems namely social insect behavior and immune systems.
– a self-control strategy based on balancing exploration and exploitation to face an evolving and unsafe environment.

Our system is viewed as an artificial ecosystem, composed of a set of active software entities called agents, that are defined with locality in mind: local perceptions, local actions. Agents evolve in the network which represents their artificial living world. The agents' community is aimed to protect its artificial living environment (the network), to provide resources for artificial living agents. An illustration of these ideas is provided in the case study (section 5) below.

4 Motivations and Related Works

In the context of large scale computer networks, motivations for self-organizing approaches is growing. Interest in such mechanisms underlies the need for characteristics such as highly distributed algorithms with local communication and decision making, self-reconfiguring, high scalability, redundancy, robustness. . . etc. In the context of a complex system such as the Internet, more and more approaches propose mechanisms relying on the combination of randomness and self-catalytic reinforcement to deal with essential information management. This is the case for the vast domain of approaches inspired by social insects behavior such as [17][7] for network load-balancing, [6][8] for network security, or [2] for information sharing in peer to peer systems. More recent approaches relying on this same schema, have been proposed for large scale distributed systems. For example in [4], a rumour routing algorithm for sensors networks is proposed. This algorithm is based on the idea of creating paths leading to each event and spreading events in the wide-network through the creation of an event flooding gradient field. A random walk exploration permits to find event paths when needed. Another work [13] proposes a novel middleware called TOTA (Tuples On The Air) for supporting adaptive context-aware activities in dynamic network scenarios. The main idea in this work is the use of a spacial environment for the representation of contextual information, and its local spreading through the notion of "computational fields" allowing adaptive interactions between the application components. We can also cite [18][16] which develop similar ideas. Through the analysis of these works, we can notice that their approaches and the approaches inspired by social insects, share the same main concept namely: balancing exploration - through random walk- and exploitation - through spreading of pertinent information using gradient fields. These approaches seems to be particularly adequate to complex networks with a scale free characteristic. Indeed, the random walk permits to reach the randomly distributed hubs (nodes with high connectivity) and the settling of gradient fields at the hub nodes because of their high degree of connectivity. This high degree permits also the reinforcement of the so constructed gradient fields. The small world feature of complex networks, is also very interesting in this kind of approaches. Their short average path length property allows for the smooth spreading of the informations represented by the gradient field over all the network. Of course, the effectiveness of such approaches depends strongly on the tuning of many parameters such as, the amount and distance of information spreading, the amount of its persistency and reinforcing rates, etc.

5 A Case Study: Network Intrusion Detection and Response

5.1 A Nature Inspired Model Enhancing Self-Organization and Self-Control

The efficiency of natural systems as complex systems completely, distributed and dynamic, was a source of foundation of our IDRS architecture. The intrusion detection system was designed with the immune system in mind whereas the intrusion response system was designed with the ant colony organization in mind. Indeed, there are some interesting parallels between these two natural systems and the IDRS:

– at a local level, both systems (natural systems as well as IDRS) are faced with complex recognition problem. More precisely, the immune system deals with self/non self protein recognition [19] to detect anomalous cells whereas the intrusion detection system deals with normal/abnormal pattern recognition to detect suspicious activities. Similarly, the social insects like ants deal with the recognition of the highest pheromonal gradient to follow the source of food, whereas the intrusion response system has to recognize remotely the type of alert launched in order to follow the source of attack [1] and perform the relevant response.
– at a global level, both systems (natural systems as well as IDRS) behave as complex systems with a set of single components engaged in local interactions. These interactions between single components exhibit a global coherent and emergent behavior. As already mentioned this self-organization process is one of the natural system properties. This property can be applied to IDRS in order to obtain an IDRS with an emergent detection and response behavior that stays safe and coherent. Globally, the detection system has to avoid as many false positive as possible whereas the immune system has to avoid so called auto-immune behavior [2]. Globally, the response system has to avoid many answers at the same location while there is no more attack, whereas an ant system has to avoid having many ants following a trail to an exhausted source of food.

5.2 A Mobile Agent Based Architecture for IDRS

We propose in this paper a distributed IDRS composed of two kinds of mobile agent population. An Intrusion Detection Agent (IDA) population taking its inspiration from immune systems and an Intrusion Response Agent (IRA) population, taking its inspiration from social insect behavior. Mobility is an essential factor in the design of our system. Indeed, as the network could be very wide, it

[1] by source of attack it is meant the location where the attack was detected. Thus, the response system independently from the detection system will be remotely activated.
[2] by auto-immune behavior happens when the immune system identifies its own safe components as unsafe or foreign ones and destroys them.

is difficult to imagine the implementation of different kinds of intrusion detection and response behavior at each node of the network. Mobility is a way to implement these behaviors at a small scale and propagate their use on a more larger scale. On another hand, having the components of the IDRS mobile, makes it more robust to attacks because of mobile agents furtivity.

Schematically the IDRS architecture is as follows:

– The considered system is a network of interconnected hosts which could be a LAN or an Intranet for instance. This Intranet is divided into several security domains depending on the topology as well as on security needs of the corresponding domain.
– A population of IDAs incarnates the Intrusion Detection System (IDS). To detect local[3] attacks, IDAs responsible for a local domain have to be able to discriminate between normal and abnormal activity. For more simplicity the good running of different programs and their deviation compared to a normal activity is supervised. For that, short sequences of system calls when the program runs in safe conditions and environment are collected, as it was done in [9]. In each local domain we run a different program and build a database specific to the program as showed in Figure 1. This avoids having too big databases needed for data collection. This is a well known problem in anomaly detection system. This also avoids having too big corresponding management. This presupposes that, at the second stage when the system runs in normal conditions, the MAs placed in each domain will be specialized in one program, in the sense that they will have to detect deviation of system calls specific to this program. In each domain, MAs specific to a program are able to memorize a safe sequence obtained from the normal profile database. Each program specific MA selects a set randomly (or selects randomly a block of n consecutive sequences) and examines locally (in each domain it is located), the deviation of the incoming sequences from the selected set (Figure 2). If the deviation is too high the MA launches an alert. Otherwise, under a certain level of deviation the sequence is accepted. Each MA can be considered as short lived because it continually selects and memorizes new sequences from its database. In order to detect anomaly emerging locally from other programs or to allow a MA specific to a program to detect an anomaly in all the subdivided Intranet the mobility is essential. Indeed each program specific agent circulates continuously and visits randomly the neighbor domains where it performs anomaly detection before returning in its home domain to do the next sequence random selection. A-priori, as the number of MAs per domain is limited, a MA is cloned before moving to the next domain, only if the level of suspicion for an anomaly becomes too high in the current domain, because the level of suspicion augments with the frequency of alerts for this anomaly.
– A population IRAs incarnates the Intrusion Response System (IRS). In parallel to the IDA population each IRA of the IRS performs a random walk

[3] Local in the sense of local domain. In fact, anomalous patterns due to distributed attacks are also considered

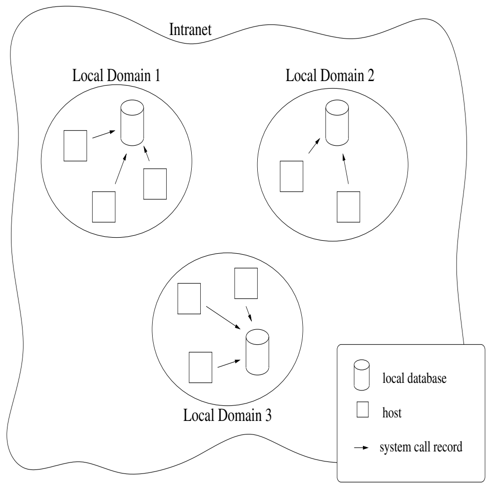


Fig. 1. The Learning Stage

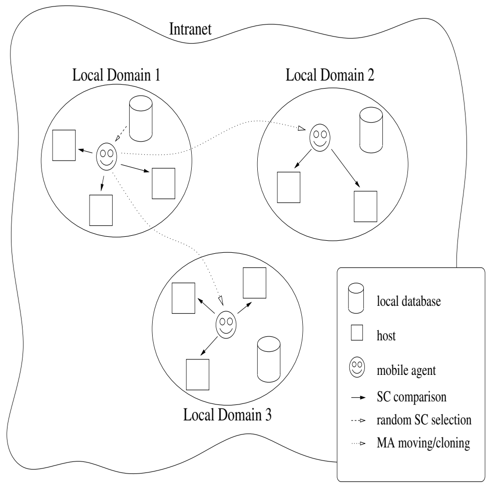


Fig. 2. The Detection Stage

through the network. As soon as an alert is launched, the IDA releases through the network a so called electronic pheromone which will be used by IRAs to trace the route back to the alert source. This electronic information embeds all the essential features dedicated to the attack. The main fields are:

- The identifier of the IDA which built the pheromone after the attack was detected.
- The pheromone creation date.

- The number of hops that corresponds to the distance in term of nodes at which the pheromone will be diffused.
- The gradient that decreases hop by hop and will be used by the IRAs to follow back the pheromonal path to the alert source.
- The suspicion index that corresponds to the importance degree of the attack. This importance degree is fixed a-priori and is adjusted during the system running.

This pheromone is diffused randomly in one direction by another agent population, namely the PheromonePropagators. This dissociation of roles is quite useful because it allows the different population of agents to work independently from each others in an asynchronous way. The pheromone is deposited and the roaming IRAs will follow it as soon as they detect its existence. Of course, this pheromonal information acts as a communication medium to alarm the good IRA population. As already said, the Intranet is logically divided into security domains and as the IDAs, the IRAs are also specialized in different tasks. Thus, the information carried by the pheromone is different according to the detected attack as well as its seriousness. This allows the IRAs to perform the adequate response. This is quite advantageous in term of performance because it avoids having inappropriate agents moving at a location where they will be useless. Moreover the pheromone will also evaporate after a certain laps of time. This laps of time is adjusted according to:

- The average time needed by an IRA to perform its task when visiting a node among all, to read and interpret the pheromonal information and also to choose the next node to visit.
- The number of IRAs that have already performed a response for the current pheromone.

Here also, this evaporation process contributes to limit IRAs activity as well as useless resources consumption.

6 Experiments and Results

This section describes the results obtained in a real network at the University of Geneva when performing a detection scenario as well as a response scenario. The combination of the mobile agent model with immune system and the social insect behavior is the driving idea behind this work. However, one cannot deliver a proof of concept unless it can be demonstrated in a real world scenario.

6.1 Detection of TCP Port Scans

The scenario that was chosen for implementation is not really an intrusion, but rather the prelude to it. Very often, an attacker will try to takeover an information system. By that term, one usually refers to gaining unauthorized administrator rights. Whether this system is the ultimate target or just an intermediate host is not very important at this time, as administrators tend not to appreciate

someone tempering with their access. One group of takeover techniques is exploits or security holes arising from bad implementation of some service running on the target host. Unchecked buffers in networking services, for example, are very frequent flaws that lead to possible exploits such as wuftp buffer overflow. At its very simplest, port scanning is exactly what it names implies. The scanner will test a number of ports of a TCP/IP node by sending packets of his choosing. This varies from ping requests that merely show whether a node is up, to protocol-incorrect sequences of TCP packets, the replies to which are usually implementation specific.
In order to detect the various types of port scans, one must analyze network traffic and interpolate common trends. An obvious signature to look for is a number of packets to different ports from the same source address. A variant of this signature consists on looking for the number of packets to different ports on different machines of the supervised network in a distributed port scan scenario. Thus, IDAs behave as sentinels and access the required data, upon which the analysis is based, through raw TCP sockets on each host. Upon screening, the sentinels select only packets that fall into the port scanning scenario described above. These sentinels own a short-term memory to register the last events of the former visited nodes. At each visited node, each sentinel correlates its own short-memory events with those present in an historic file of the node. In fact, this historic file incarnates a long-term memory and is nothing else than the trace deposited in the current node by all the other sentinels. In this way, IDAs behave as simple sentinels interacting with one another through a common shared environment; here through the different network nodes. Thus, IDAs are collectively able to exhibit a complex global detection scenario, the distributed port scan, thanks to these local interactions.

Measured Results

– The port scan detection
  The IDS, as described in the previous section, behaves, from a summary point of view, in the following way. A given sentinel moves randomly between nodes that participate in the IDS. On each node it scans TCP network traffic and retains the source and time of suspicious packets. It stores it in memory and delivers alerts when the recorded data indicate short intervals between a large number of packets from the same source.
  The main variables of the sentinel's algorithm are what limit values it allows for these measures. The threshold for minimum average intervals between suspicious packets has an initial value and is adjusted downwards as more and more packets are detected as showed in Figure 3. It also has a linear inertia factor that pushes it back to its initial state over long periods of inactivity. Thus, the threshold is a combined measure of both number of packets and intervals between them. Tests were conducted on various configurations to quantify these variables. Presented here are the results collected when executing the IDS on 15 virtual machines running Linux inside an IBM S/390 mainframe (Figure 4). The entire IDRS was implemented using Sun

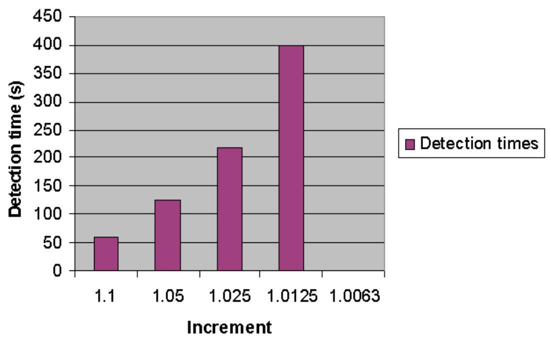


Fig. 3. Time needed to set off the alarm. The X-axis values indicate different values of the increment of the threshold when new suspicious packets are recorded. This is the primary parameter to set when customizing for a particular environment

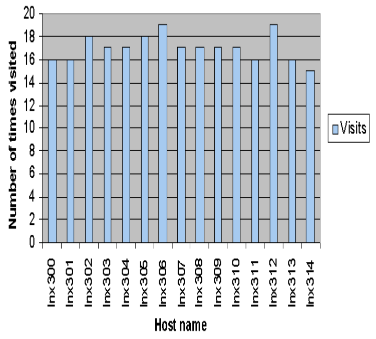


Fig. 4. IDA Visit Pattern. The sentinel moves around nodes without affinity to any

Microsystems Java technology and Coco Software Engineering JSeal-2 [11] framework. The latter is a mobile agent platform itself implemented in pure Java.
At the beginning (Figure 5) the sentinel threshold value is low (red curve) because the average interval between packets measured by the sentinel is high (green curve). Gradually, during its random walk the sentinel correlates the different scan information deposited by other sentinels in the current visited node (memory plus historic information). In average as soon as the interval between packets becomes to low during a lapse of time (the green curve decreases), the sentinel's stress increases. And the red curve augments too. When curves meet, alert is given. The sentinel then escapes the node where the alerts was launched after generating the pheromone and it sets

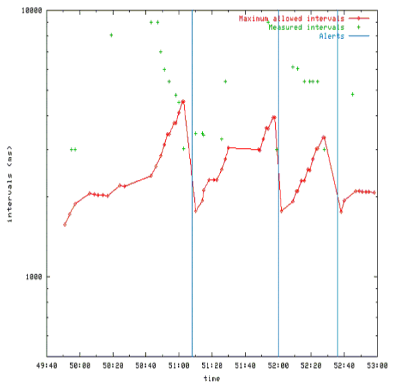


Fig. 5. Threshold, suspicion relationship for a continuous long-term port scan. The Y-axis represents the average interval between packets as measured by the sentinel throughout the test execution. The alternate curve indicates the threshold value for this variable. X-axis represents the time

Table 1. Response frequency according to the number of hops. The field success indicates the number of IRAs which answered in time

Number of hops	Mean time to source [sec]	Success
7	14.60	100 %
3	17.20	90 %
2	8.14	70 %

the threshold to its initial value. Then, the response scenario can take place.

– The port scan answer
After an alert, the corresponding pheromone is generated, awaiting the response mechanism to come into play. The response mechanism is straightforward in this case. IRAs trace back the source of the attack and switch to an aggressive mode which allows them to analyze more precisely the entering packets before reacting. They have the possibility depending on the service under scrutiny, to refer to a knowledge database of vulnerabilities. This database provides information on the more frequent as well as the new vulnerabilities. For instance, one well known exploit is wuftp buffer overflow. This exploit has been performed after the ftp service was scanned on the IBM virtual machines. The IRAs role was to stop the ftp service on the corresponding virtual machines. Thus, Table 1 shows the response frequency according to the number of hops in the case of an ftp port scan scenario.

6.2 IDRS Self-Organization Behavior

In consequence of the results obtained with the implementation, the IDRS self-organized nature is highlighted. Indeed, as mentioned in section 3.2, the IDRS exhibits the two self-control principles which are:

– Exploitation
  IDAs exploit the deposited information (average interval between packets) at each intermediate node thanks to the node history. Besides IDAs influence the node history and in this manner the other IDAs, thanks to their short-term memory. Thus, they evolve in a self catalytic process to do network control. In fact, their stress state depends on this self-catalytic information exploitation. Figure 5 illustrates completely this effect; IDAs stress increases until the alert is launched. After the alert, the memory is cleaned and the stress decreases since there is no more effect.

– Exploration
  IRAs explore the deposited pheromone at each visited node thanks to the diffusion process. This exploration allows them to find new possibilities to reach the source of alerts. Table 1 illustrates this exploration effect.The higher the diffusion distance (the higher the exploration effect), the higher the percentage of success.

7 Conclusion and Future Works

In this contribution, we have presented the use of a complex system perspective to develop networks applications. We argue that a network of computers exhibits many of the characteristics of complex systems and propose to adopt a nature inspired paradigm to develop network oriented applications. We illustrate our proposal by describing a distributed system for intrusion detection and response. In this context, the network of computers is represented as an artificial ecosystem, where artificial creatures implemented using mobile agent technology, collaborate to detect intrusions and launch an alert to make the system (or administrator) respond to the detected attack.

The leading idea in this paper was to show how self-organizing behaviors such as those exhibited by social insects or immune systems provide a good perspective to implement self-organizing distributed systems evolving in complex environment such as networks of computers.

In this perspective, this work is a beginning. Indeed, a real theory need to be developed to tackle the issues of engineering network applications following the perspective of complex systems. Many efforts are engaged in this direction by many researchers, and new results are provided continuously, which lets think that this perspective is a very promising one.

Acknowledgements

Thanks to Edwin Cox and Alexandre Nevski for their contribution to the implementation.

References

[1] R. Albert, H. Jeong, and A. Barabasi. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000. 125

[2] O. Babaoglu, H. Meling, and A. Montresor. Anthill: A framework for the development of agent-based paeer to peer systems. In *Proceedings of the ICDCS'02*, Vienna, A., July 2002. 128

[3] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999. 124, 125, 126

[4] D. Braginsky and D. Estrin. Rumour routing algorithm for sensor networks. In *Proceedings of the Fisrt Workshop on Sensor Networks and Applications (WSNA)*, Atlanta,GA, USA., September 2002. 128

[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the ACM SIGCOMM'99, Cambridge, MA, USA*, pages 251–262, 1999. 124, 125

[6] S. Fenet and S. Hassas. A distributed intrusion detection and response system based on mobile autonomous agents using social insects communication. *Electronic Notes in Theoretical Computer Science*, 63:21–31, 2002. 128

[7] S. Fenet and S. Hassas. An ant based system for dynamic multiple criteria balancing. In *Proceedings of the Fisrt Workshop on ANT Systems*, Brussels, Belgium, September 1998. 128

[8] N. Foukia, S. Hassas, S. Fenet, and P. Albuquerque. Combining immune systems and social insect metaphors: a paradigm for distributed intrusion detection and response systems. In *Proceedings of the 5th International Workshop on Mobile Agents for Telecommunication Applications, MATA'03*, Marrakech, Morocco, October 2003. Lecture Notes in Computer Science -Springer Verlag. 128

[9] S. Hofmeyr and S. Forrest. Architecture for an artificial immune system. *Evolutionary Computation 7(1), Morgan-Kaufmann, San Francisco, CA, pp. 1289-1296*, 2000. 130

[10] J. H. Holland. Adaptation in natural and artificial systems. *MIT Press, Cambridge, MA*, 1992. 126

[11] J-Seal2. `http://www.coco.co.at/development/`. 134

[12] M. Mitchell. Analogy-making as a complex adaptive system. *L. Segel and I. Cohen (editors), Design Principles for the Immune System and Other Distributed Autonomous Systems. New York: Oxford University Press*, 2000. 127

[13] M.Mamei, F. Zambonelli, and L. Leonardi. Tuples on the air: a middleware for context-aware computing in dynamic networks. In *Proceedings of the Fisrt International ICDCS Workshop on Mobile Computing Middleware (MCM03)*, Providence, Rhode Island., May 2002. 128

[14] S. T. Park, A. Khrabrov, D. M. Pennock, S. Lawrence, C. L. Giles, and L. H. Ungar. Static and dynamic analysis of the internet's susceptibility to faults and attacks. *IEEE INFOCOM*, 2003. 125

[15] R.Albert and A.-L.Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics 74.*, 2001. 124, 125

[16] R.Menezes and R. Tolksdorf. A new approach to scalable linda-systems based on swarms. In *Proceedings of the 18th ACM Symposium on Applied Computing (SAC 2003)*, 2003. 128

[17] R.Schoonderwoerd, O. Holland, and J.Bruten. Ant-like agents for load balancing in telecommunications networks. In *Proceedings of the 1st International Conference on Autonomous Agents*, pages 209–216, February 5-8 1997. 128

[18] S.Bandini and S. Manzoniand C. Simone. Heterogenous agents situated in heterogenous spaces. In *3rd International Symposium From Agent Theories to Agent Implementations*, Vienna, A., April 2002. 128

[19] Alfred Tauber. *The Biological Notion of Self and Non-self*. The Stanford Encyclopedia of Philosophy (Summer 2002 Edition), Edward N. Zalta (ed.), `http://plato.stanford.edu/archives/sum2002/entries/biology-self/`, 2002. 129

[20] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, 1998. 125

A Holonic Self-Organization Approach to the Design of Emergent e-Logistics Infrastructures

Mihaela Ulieru[1] and Rainer Unland[2]

[1] Electrical and Computer Engineering Department, The University of Calgary, Calgary
T2N 1N4, Alberta, Canada
ulieru@ucalgary.ca
http://isg.enme.ucalgary.ca
2 Institute for Computer Science and Busines Information Systems (ICB)
University of Duisburg-Essen
45117 Essen, Germany
Unlandr@cs.uni-essen.de
http://www.cs.uni-essen.de/dawis/

Abstract. This paper presents first results for a theoretical foundation for emergency e-Logistics based on an extended FIPA-compliant layered multi-agent system architecture. The approach is based on the holonic paradigm proposed by A. Koestler in his attempt to create a model for self-organization in biological systems and proven by the HMS Consortium to be very useful for resource management and allocation in the manufacturing domain. We integrate the mechanism of emergence into the holonic paradigm encompassing both vertical and horizontal integration of resources in a distributed organization to enable the dynamic creation, refinement and optimization of flexible ad-hoc infrastructures for emergency e-logistics management.

1 Introduction and Project Objectives

This work originates in previous results targeting the development of enabling information infrastructures for global production integration obtained working with the Holonic Manufacturing Systems (HMS) Consortium in the past seven years [22, 39, 29, 24]. In particular, within the Holonic Enterprise Project [38] joining international research and industrial forces in a unified effort to develop integration FIPA-compliant standards for global collaborative enterprises, within the Project FIPA-Enabled Supply Chain Management [32, 31, 30, 23] a 1-Tier collaborative agents architecture for customer-centric global supply chain was proposed. This is currently being extended to the implementation and standardization of Holonic Enterprises [33, 36]. Recently a theoretical model inducing *emergence* in holonic virtual organizations has been developed [35, 37], which expands previous results in the modeling of multi-agent systems dynamics [29, 26, 28] with a mechanism of evolutionary search in Cyberspace [27].

The *overall objective* of this work is a theoretical foundation and the definition of *standards for emergency e-Logistics based on an extended FIPA-compliant layered multi-agent system architecture.* We target automated mechanisms for ad-hoc

© Springer-Verlag Berlin Heidelberg 2004

creation of collaborative organizational structures endowed with highly efficient and cost-effective resource allocation and management for disaster relief, medical or military emergency applications, capable to constantly optimize and adapt their structure and behavior according to new/emerging needs. Addressing and handling management mechanisms for emergency logistics is a natural development following our existing results in the field of holonic agency. Such a mechanism would ensure as well customer-centric logistics management in manufacturing supply chain management and other application areas. We aim to expand the emergence model presented in [35] to a layered (3-tier) holonic architecture, thus building a cross-level (vertical integration)/cross-organizational (horizontal integration) coordination backbone for fast reactivity, dynamicity, and flexibility.

In the next section we will discuss state-of-the-art in e-logistics infrastructures. Section 3 introduces our emergent e-Logistics infrastructure model. It is based on a layered multi-agent system architecture. We will especially discuss two important issues, namely, how cross-enterprise services can be discovered and selected (section 3.4) and how agents can be flexibly equipped with different kinds of goals that allow them to react in different ways to different requirements (section 3.5). In section (section 3.2 an example scenario is introduced which provides the context for the discussion of the above issues. Finally, section 4 concludes the paper.

2 State-of-the-Art in the Development of e-Logistics Infrastructures Supporting Emergency Response Management

In today's dramatic context lessons learned in other areas are transferred and adapted to leverage the quick development of response mechanisms to emergency situations [18]. Private sector "best practices", in particular techniques inherited from enterprise logistics and supply chain management, are considered especially useful in addressing emergency preparedness planning and defense logistics as well as post-incident analysis and reporting [4].

Major Research Consortia, such as the EU (IST-1999-21000) MobiCom, recently calibrated their purpose to respond to the new demands with adequate technologies [8]. The e-Motion project of the MobiCom Consortium aims to enhance citizen safety and security by proving the interoperability of diverse wireless technologies and their mutual support for a better response to emergency cases through three trial set-ups: Emergency Management, Tele-ambulance, and Transport Surveillance [6]. Although they provide a first big step towards emergency response handling by ensuring the seamless integration of devices/units/entities involved (Fig. 1, from [6]), the second big step - development of collaborative coordination infrastructures enabling (vertical and horizontal) workflow management in virtual organizations [33], has not yet been considered in the deployment of emergency logistics operations. Rather a flat approach ensuring peer-to-peer communication of organizations and devices is taken [6]. This approach characterizes as well most literature related to virtual organizations creation, where coalition formation [16] addresses mostly the horizontal integration of several organizations through various brokerage mechanisms ([11, 14]).

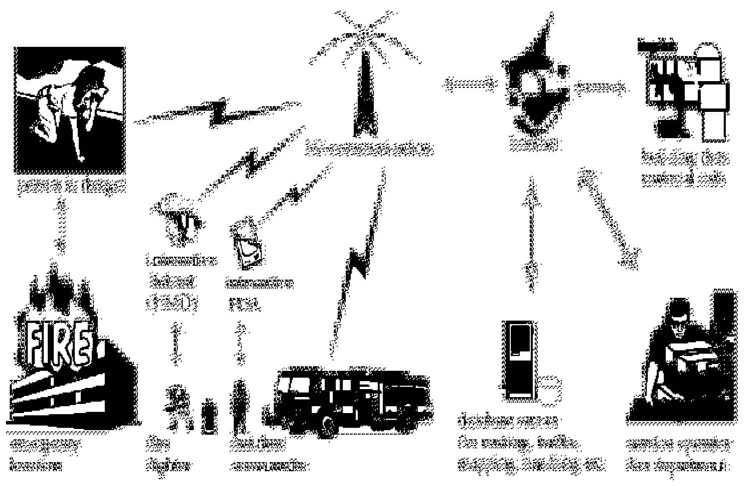


Fig. 1. Emergency Response Scenario

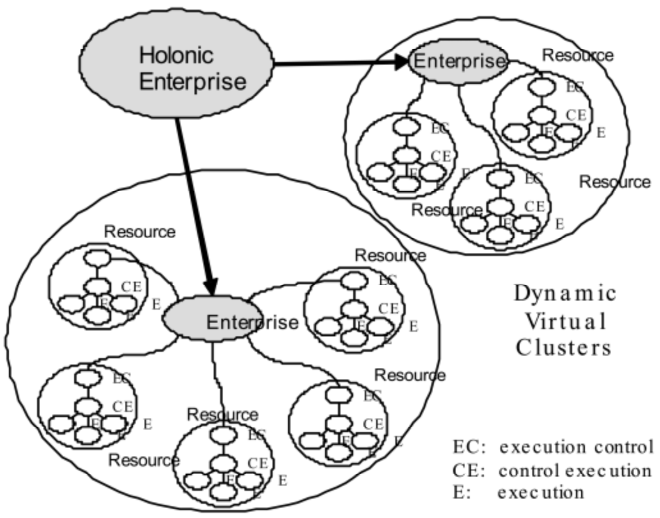


Fig. 2. Holonic Enterprise

More advanced approaches to virtual organization integration [21] propose a 3-tier cross-organizational workflow model that captures private partner workflows, workflow-views and coalition workflows. By this they overcome the limitations of the 1-tier peer-to-peer [11] or 2-tier private-public models [9]. Although such architectures respond to the needs of dynamic "on-the-fly" resource allocation (enabling the self-organization) they are limited (as well as the other models [11, 9, 14, 16]) to a predefined/static structure of a virtual organization with fixed/known partners and resources. State-of-the-art in "key-to-glass" [1] service discovery and composition [3, 19] are not exploited to create, refine and optimize ad-hoc context based [43] and purpose-driven dynamic organizational structures – the most challenging requirement in emergency logistics management [41].

To date techniques for on-demand user needs [7] using nested interchangeable views suggest that nested hierarchies (holarchies [37]) may appropriately model coordination of activities among “first responders” at all levels in a confederated structure by enabling federal government emergency response plans to mesh with the regional, state and local counterparts [18]. Visionary companies, such as TheMindElectric, are already looking into the creation of dynamic service infrastructures to support emergence of architectures in Cyberspace that mirror biological behavior.

3 An Emergent e-Logistics Infrastructure Model Based on a Layered Multi-Agent System Architecture

We start from the observation that the phenomenon of emergence involves:

- *Self-organization* of the dynamical systems such that the synergetic effects can occur
- Interaction with other systems from which the synergetic properties can *evolve*

We build on a recently developed fuzzy-evolutionary model [35] that induces emergence in virtual societies of software agents “living” in Cyberspace:

- By minimizing the entropy measuring the (fuzzy) information spread across the multi-agent system it induces *self-organization* with careful clustering of resources (agents) ensuring a good interaction between the system’s parts to reach its objectives timely, efficiently and effectively.

It enables the system’s *evolution* by interaction with external systems (agents) e.g. found via genetic search in cyberspace (mimicking mating with most fit partners in natural evolution) or via the use of dynamic discovery services [3, 14, 19] (yellow pages, mediators/brokers, etc.).

The integration of the two above mentioned steps of the formalism on a flat, 1-tier (peer-to-peer) architecture via the definition of the fitness function in terms of fuzzy entropy is addressed in [37]. Here we present the extension of the emergence mechanism from a 1-tier to a 3-tier nested (holonic) architecture with intra-enterprise (planning) and physical resource levels, Fig. 2.

Multi-agent systems enable cloning of real-life systems into autonomous software entities with a “life” of their own in the dynamic information environment offered by today’s cyberspace. The *holonic enterprise* has emerged as a business paradigm from the need for flexible open reconfigurable models capable to emulate the market dynamics in the networked economy (cf. [12]), which necessitates that strategies and relationships evolve over time, changing with the dynamic business environment. Building on the MAS-Internet-Soft Computing triad to create a web-centric model endowing virtual communities/societies (generically coined as “enterprises”) with proactive self-organizing properties in an open environment connected via the dynamic Web, the holonic enterprise paradigm provides a framework for information and resource management in global virtual enterprises (cf. [35]).

3.1 Architecture

According to the above paradigm of holonic (virtual) enterprises, which - from our point of view - is the most sophisticated approach to virtual enterprises, the following three principle layers of collaboration/interaction need to be considered by a MAS in order to emulate a holonic enterprise.

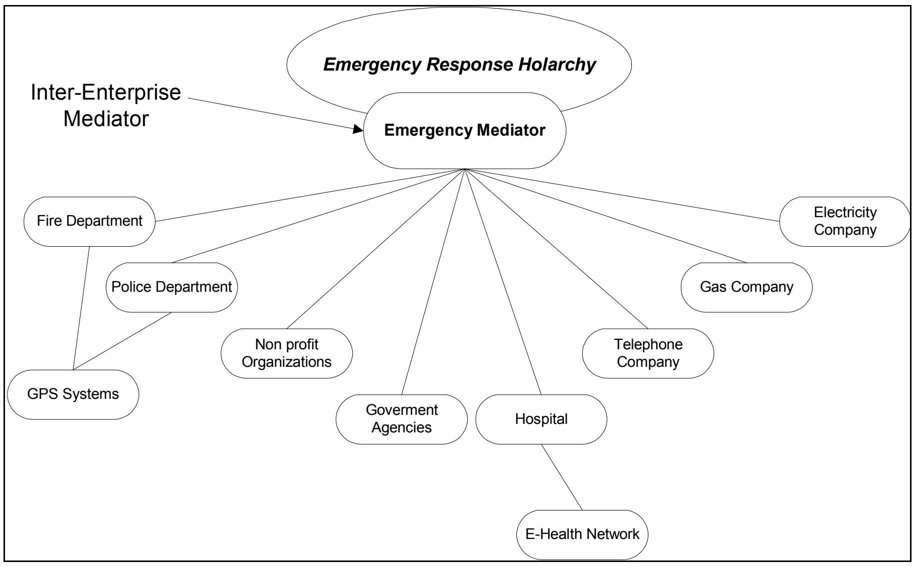


Fig. 3. Inter-enterprise level for the scenario in Fig. 1

Inter-Enterprise Level

The highest level, the *Global Inter-Enterprise Collaborative Level* or *Inter-Enterprise Level* for short, is the level on which the holonic enterprise is formed. Fig. 3 depicts the inter-enterprise level for the scenario presented in Fig. 1. Each collaborative partner is modeled as an agent that encapsulates those abstractions relevant to the particular cooperation. By this, a dynamic virtual cluster emerges that on the one hand is supposed to satisfy the complex overall goal at hand as good as possible and on the other hand considers the individual goals of each enterprise involved.

Intra-Enterprise Level

Before an enterprise can undertake responsibility for some subtask, it has to find out about its own internal resources to ensure that it can deliver on time according to the coordination requirements of the collaborative cluster and with the required quality. Since this is done within the enterprise at hand we call this level the *Intra-Enterprise*

Level. Fig. 4 depicts the intra-enterprise level for one of the holons in Fig. 1. the fire department holarchy.

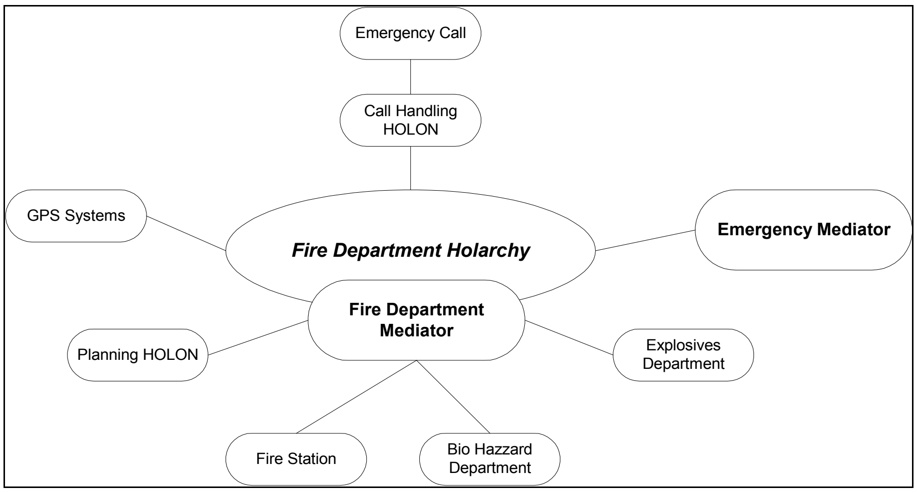


Fig. 4. The Fire Department Holarchy (intra-enterprise level for Fig. 1 scenario)

Atomic Autonomous Systems or Machine Level

The lowest level is the *atomic autonomous systems* or *machine level*. It is concerned with the distributed control of the physical machines that actually perform the work. To enable agile manufacturing through the deployment of self-reconfiguring, intelligent distributed automation elements each machine is cloned as an agent that abstracts those parameters needed for the configuration of the virtual control system managing the distributed production.

Each level is represented by one or a number of holons each of which representing a unit on this level. Such a unit recursively consists of a set of holons each of which being responsible for a subtask of the overall task of the holon/holarchy at hand. These sub- holons do not need to be from lower levels only, but can as well be holons from the same level. For example, a flexible cell holon may comprise holons representing atomic autonomous systems as well as flexible cells which perform tasks that are necessary for the flexible cell holon at hand to be effective.

Planning and dynamic scheduling of resources on all these levels enable functional reconfiguration and flexibility via (re)selecting functional units, (re)assigning their locations, and (re)defining their interconnections (e.g., rerouting around a broken machine, changing the functions of a multi-functional machine). This is achieved through a replication of the dynamic virtual clustering mechanism having now each resource within the enterprise cloned as an agent that abstracts those functional characteristics relevant to the specific task assigned by the collaborative conglomerate to the partner. Re-configuration of schedules to cope with new orders or unexpected disturbances (e.g. when a machine breaks) is enabled through re-clustering of the agents representing the actual resources of the enterprise. The main criteria for resource (re)allocation when (re)configuring the schedules are related to cost minimization achieved via multi-criteria optimization.

Mediator Holon/Agent

The basic condition for holonic systems is that a holon is simultaneously a “whole” and a “part” of some other whole/holon. Thus the system exhibits a tree-like structure. Each inner node of this tree in fact comprises a set of (sub-)holons that cooperatively work together to solve the specific task that is assigned to this node. In order to communicate with such a set of (sub-)holons one need to define a representative for it. In a MAS environment this representative can be a software agent that undertakes the task of a mediator. It fulfills two main functions. First, seen from the *outside*, it acts as the interface between the agents inside the holon and those outside it; conceptually, it constitutes the agent that represents the holon. Second, seen from the *inside*, it may initiate and supervise the interactions between the group of sub-holons/agents within the holon at hand; this also allows the system architect to implement (and later update) a variety of forms of interaction easily and effectively, thereby fulfilling the need for flexibility and re-configurability. The mediator encapsulates the mechanism that clusters the holons into collaborative groups. The architectural structure in such holarchies follows the design principles for metamorphic architectures.

From the point of view of the mediator agent the first case describes its *outside view* while the second case describes its *inside view*. Both views are significantly different with the outside view usually being much more demanding than the inside view.

Since the mediator is the common representative for the outside world of the holon it represents, as well as for the inside world, it needs to understand both worlds. For this, two kinds of ontologies are necessary, namely for “peer-to-peer” communication at each level (that is “inter-agent” communication among entities that form a cluster); and for “inter-level” communication that enables deployment of tasks assigned at higher levels (by the mediator) on lower level clusters of resources.

3.2 Example Scenario

Let us come back to the Fig. 1 scenario. In case of a catastrophe all possible measures have to be taken in order to keep its effects (especially causalities) as small as possible. Profit in the sense of monetary benefits is not the main goal. Instead, a fast reaction is required. First of all it has to be ensured that all possible resources are allocated as fast as possible. In a next (refinement) step a good coordination of the resources is necessary. Lets have a look at the subtask ambulance cars coordination. In a first step it is necessary to ask all of them to drive as quickly as possible to the place where the catastrophe has happened. If necessary other cars that may also be equipped for transporting injured persons may be asked to help. In the next step it needs to be coordinated which and what kind of patients are driven to what hospital. This depends on the capacity of the hospital as well as its specialty and equipment. In these extreme situations the otherwise important system goal that the ambulance cars of hospital X bring the patients only to this hospital has to be suspended and replaced by a strategy which allows optimal distribution of the patients independently of what cars are transporting them.

With respect to our multi-agent system based emergent e-Logistics infrastructure the above requirements mean that the agent being in charge of the transportation and treatment of the injured people (called ambulance agent from now on) needs to react very quickly in the beginning to start the rescue action and then, on the fly, has to start to more and more optimize the originally pretty abstract and general coordination respectively planning. This requires a flexible adaptation of the overall goals of the agent and their preferences/priorities. After the initial phase of extreme fast responses (which can be translated to an approach closer to pattern matching) the further refinement of the planning procedure needs to consider more and more the real circumstances of the given scenario, therefore, needs to shift from a more reactive to a more deliberative behavior. Police cars need to be sent to the right places in order to make sure that the ambulance cars can drive as fast as possible to the designated hospitals. The personal in the ambulance cars (which can be considered as holons on the lowest level as well) need to diagnose their patients in order to provide a basis for the decision which hospital may be the right one. Then, on the next higher level, it needs to be decided which hospital is effectively chosen (otherwise, if the ambulance car decides, it can happen that too many ambulance cars are driving to the same hospital, thus, overstrain its real-time capacity or the capacity of the streets leading to it). This requires that the ambulance agent defines tasks and delegates them to special agents which can deal with them exclusively. All of this has to be done in a step by step refinement process which needs to consider all the influences by all the other measures that are to be taken to deal with the catastrophe (e.g. the coordination of other kinds of vehicles that are needed to get access to injured people or, e.g., to extinguish fire or avoid environmental catastrophes (oil leakage)).

In this paper we will deal with several aspects of the above scenario. First of all, we will discuss what kinds of mediator agents are needed in an emergent holonic enterprise since the above observation requires designing mediators on different levels differently (section 3.4). Next we will address the problem how different kinds of goals can be deeply embedded in an agents belief system and how the belief system can be modeled in a way that it can react in different ways (faster more automatic (reactive) response versus deliberate, however, slower response; section 0).

3.3 Expression of Semantics

One main obstacle to the meaningful interoperation and mediation of services is the syntactic and semantic heterogeneity of data and knowledge the mediator agent does access and receive from multiple heterogeneous agents (cf. [20]). The functional capability of the agent to resolve such heterogeneities refers to the knowledge-based process of *semantic brokering*. Most methods to resolve semantic heterogeneities rely on using partial or global ontological knowledge[1], which is to be shared among the

[1] An *ontology* is a computer-readable representation of the real world in form of objects, concepts, and relationships.

agents. This requires a mediator agent to provide some kind of ontology services for statically or dynamically creating, loading, managing, and appropriately using given domain-specific or common-sense ontologies as well as inter-ontology relations when it wants to communicate and negotiate with different agents or understand complex tasks.

In fact, a mediator agent that is looking for a specific task (service provider) has to parse, understand and validate the offers/service descriptions it gets. This is in order to efficiently determine which of the services and capabilities of other enterprise/holon agents are most appropriate. Typically for this ontology services are used as well. Depending on the complexity and required quality of the service a matching process may rely on simple keyword and value matching, use of data structure and type inferences, and/or the use of rather complex reasoning mechanisms such as concept subsumption and finite constraint matching. Semantically meaningful matching requires the matching service to be strongly interrelated particularly with the class of services enabling semantic interoperation between agents.

In emergency logistics, where the scope of possible organizations/tasks/skills is not restricted and/or predefined, it is difficult to express and code enough real world semantics to permit a goal-driven and effective communication between levels. We are fully aware that the expression and understanding of semantics is still a hot topic in research with sufficient general solutions even not being on the horizon. However, this does not prevent us from deploying our concepts in more restricted and better addressable application areas.

Therefore, our current approach is to use for the expression of semantics and for proper communication between levels within and across organizations already developed real-world knowledge models and ontologies [11, 21, 14, 34, 33] like DAML-OIL, respectively it successor DAML-S. Since FIPA is working on concepts to integrate agents and Web-Services we will adapt their solution to our scenario as soon as the results are available.

3.4 Discovery, Selection and Assembly (Workflow) of Cross-Enterprise Services

In order for the emergent e-Logistics Infrastructure to be operational we first of all need to develop a mechanism enabling the discovery of new partners/collaborators in Cyberspace, by using state-of-the-art approaches for coalition formation (acquaintance models) and dynamic service discovery and composition [1, 2, 3, 7, 14, 17] by a mediator agent that has to fulfill (a subset of) the following tasks:

- to understand and interpret the complex task at hand (overall objective of the holonic enterprise)
- to decompose it into sub-tasks that can be dealt with by individual enterprises
- searching for and finding possible partners
- organizing the negotiation process and dealing with possible contractors/participants
- controlling and supervising the execution of the (sub-)tasks
- reacting to and solving emerging (unexpected) problems storing and maintaining a knowledge base in which experiences of the performance and efficiency of past holonic enterprises and its members are stored

These are very special abilities by which the mediator agent is to be equipped. In the following we will first discuss the principle ways in which a mediator may search for/find the appropriate partners.

Types of Mediator Agents in a Holonic Enterprise

In principle a mediator may engage a broker/facilitator, a matchmaker, may use yellow page services or may rely (totally or partially) on its acquaintances knowledge base.

If a (mediator) agent that is looking for some service uses a *broker service* it gives up control about the selection process and about the knowledge in what way which enterprise has contributed to the solution process. It fully depends on the quality and sincerity of the broker service. *Matchmaking* already allows the mediator agent to keep much more control about the choice of the participating agents since the mediator agent only gets a list of enterprises that are in principle capable to provide some service for the solution process. However, it is left to the mediator agent to decide on what service/enterprise is to be chosen. The most independent approach is that an agent maintains its own database of *acquaintances* in which all possible cooperation partners along with their capabilities and past experiences with them are stored. The agent will always first try to find appropriate partner agents in this database. Only if this is not successful it will deploy one of the other possible services. While this approach promises the most independence of an agent from its environment it also requires a pretty sophisticated agent architecture since the agent needs to fully understand the overall goals and requirements of the requests it is supposed to find a solution for.

Of course, in case of emergencies (machine break-down) or simply because a needed capacity/service/capability is not available even on the intra-enterprise level the necessity may arise to publicly advertise a task/service in the outside world. However, in this case the mediator may make use of a broker in order to settle and process the transaction at hand. The broker can, of course, as well be the mediator agent representing the enterprise at hand.

As we move down the layers of a holonic enterprise we can observe a decrease with respect to the following features:

Time Scale

From top to bottom time scales become shorter and real-time constraints change from soft to hard real-time. While, e.g., a formation process of a new holonic enterprise on the inter-enterprise level may take a longer time and time pressures are only soft (more deliberative style of agent) an agent that represents a tool needs to usually react extremely fast (more reactive style of agent that reacts directly to patterns (□prefabricated□reaction)).

Complexity/Sophistication

From top to bottom the degree of agency decreases. The higher level agents are more sophisticated but slower, while lower agents are fast and light-weight. For example, the mediator agent that represents an enterprise needs to have a very broad knowledge and a huge number of sophisticated skills. On the contrary, the agent on the level of an atomic system just needs to have an understanding of this system together with some basic skills to communicate with the next higher level agents.

Autonomy
Since each agent will act according to its own goals conflicting situations may occur when a contacted agent does not want to contribute (fully) to the task at hand (due to overload, preference/priority given to more lucrative offers, etc.). In this case the superordinated (higher-level) agent has to decide on how to deal with such obstacles and, if possible, to develop an alternative solution on the basis of all the information obtained from all its subordinated agents (using its knowledge base and goals). Especially, in emergency cases it may be necessary to compromise the autonomy of agents completely. Again, agents that represent an enterprise will be fully autonomous and, therefore, cannot be forced from the outside to do something they do not want to do. On the intra-enterprise level the autonomy will decline with decreasing level. If, e.g., a task was accepted for which a high contract penalty is to be paid in case it is not finished in time the holon in charge may decide that the subordinate holons have to execute some task first in order not to risk a delay of the high-priority task, even if the lower level holons are already fully booked with other tasks.

The above observation requires designing mediators on different levels differently. We will come back to this topic in the next section when we will discuss how outside goals (that are imposed on an agent from the outside world) and inside goals (the goals of an agent) can be harmoniously integrated.

3.5 Intentional Problem Solving

To realize a layered three-tier holonic enterprise (Fig. 2), the usual capability of individual agents to express and represent their own goals has to be enhanced with an integration mechanism synchronizing individual goals with those of the higher levels in the extended enterprise. More specifically, this section will discuss what kind of goals need to be deeply embedded in a (mediator) agent's belief system in order for it to be capable to truly represent its underlying holon on the one hand and consider the goals of the environment it is embedded in on the other hand and how this can be realized.

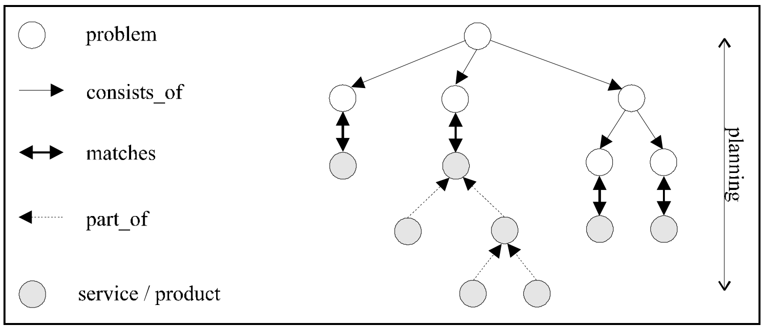


Fig. 5. Planning

We will concentrate on a single organizational unit and its (problem solving) behavior. Such a unit can be a profit-center (on the level of production planning, shop floor or flexible cell) or an individual corporation. In order to provide complex products or service bundles, first of all, a comprehensive planning process is to be performed. To achieve flexibility and dynamicity the process needs to rely on the concept of modularized capabilities. Thus planning enables the identification of those elementary products or services that can be combined for product or service bundles. As is illustrated by Fig. 3, planning can be performed *top-down*, that is, by problem decomposition[2], or *bottom-up*, that is, by service or product aggregation.

To find *the* optimal solution for such a kind of planning is in general an NP-hard problem. By exploiting heuristics/experiences we aim to overcome the limitations of existing approaches [14, 16, 19], especially regarding the timely response constraint required by emergency.

In general, during a planning process, from the perspective of the organizational unit that alternative has to be chosen that promises the highest profit[3]. Therefore, the organizational unit has to open up its planning for this profit dimension in order to behave (economically) reasonable. Consequently, two different kinds of goals can be distinguished and described: *output* and *system* goal. According to [15] the former stands for the plan goal that is to be achieved, that is the problem that has to be solved. The latter describes the real purpose of the organizational unit, which often means profit maximization.

An agent that represents such a unit must consider both goals. Systems showing a behavior that is dependent on certain goals in a reasonable way are characterized as intentional [5]. Intentionality is a key characteristic of agent-based technology. In addition agent methodology genuinely addresses planning respectively deliberative behavior [42] and cooperation.

The coexistence of two kinds of goals that have both to be considered by the generated plan necessitates a two-staged planning process. Due to the overriding importance of the system goal in comparison to the output goal the traditional planning process that mainly concentrates on the realization of the output goal has to be revised in a way that it reflects the system goals in an appropriate way. However, system goals, to become effective, must strongly influence the planning process as such. Planning processes can best be influenced if they are controlled by meta-planning processes, that is, processes on a more abstract level. In such an architecture the basic problem solving behavior of an agent can be illustrated as in Fig. 4. It visualizes the interdependencies between the most important entities involved.

An in-depth discussion of both planning and meta-planning procedures is beyond the scope of this paper, especially since traditional planning has already been intensively studied in literature. Instead, we will only concentrate on meta-planning. Meta-Planning has to ensure that, based on the system goals, a rational choice can be

[2] It is important to understand that the decompositions have to completely cover the original problem. If, as a result of the decomposition process, a subsequent synthesis is required, it has to be encoded in one of the sub-problems.

[3] In this section we will use the word profit as a simplified expression for the overall goal of a holonic unit. Profit has to be interpreted in a very broad sense. It may mean monetary profit as well as to save lifes by reacting as quickly as possible without considering possible costs.

made among planning actions that constitute alternatives[4] with respect to the given output goal. Consequently meta-planning has to control the planning process either in a dynamic or in a static manner:

- *Dynamic* meta-planning stands for the direct control of the planning procedure. This means that whenever the planning process has to make a choice among alternative decompositions or, correspondingly, among alternative products or services the meta-planning process will determine the one from which it assumes that it is the most appropriate one with respect to the system goals.
- *Static* meta-planning describes the procedure where first of all a number of relevant alternative plans are generated. Subsequently that plan is chosen that reflects the system goals best. This corresponds to an extensive, however, non exhaustive[5] search through the set of all possible plans. This set represents the overall search space and is obviously relatively complex.

Both alternatives operate analogously on two subsequent levels of aggregation: Depending on the respective system goal both select one element from a set of alternatives. Dynamic meta-planning does so for a set of decompositions respectively services or products whereas its static counterpart works on a set of plans that again constitute a set of decompositions and products or services. Since the system goal has to enable a total ordering of the alternative elements it has to be defined by referencing some of their characteristics (e.g., profit contribution of each product). Consequently, system goals can be distinguished according to the kind of element they refer to. Thus dynamic meta-planning requires a system goal that is located at a decomposition / product or service level and static meta-planning necessitates system goals that are situated at plan level. By continuing analogously the aggregation by combining alternative plans we get to a system goal at a plan-set level.

Within the scenario of this paper such system goals can be formulated as follows:

Decomposition / product or service level:
"Make as much profit as possible by preferring certain products/services."

Plan level:
"Make as much profit as possible from each task respectively problem."

Plan-set level:
"Make as much profit as possible within a certain period of time."

Achieving a system goal at the plan-set level would constitute an "ex-post" meta-planning and can consequently not be implemented directly, because a runtime

[4] This choice among alternatives does not comprise all choices that are available to the planning procedure. Alternative in this context means that the respective products or services are likewise applicable within a given planning stage. Without meta-planning the planning procedure would use heuristics to decide on one of them, because from a planning perspective they equally serve the output goal. Otherwise meta-planning would actually comprise planning.

[5] An exhaustive search would be NP-hard.

control of the planning procedure can only be realized by applying either dynamic or static meta-planning. However, both require system goals located at "lower" levels, i.e., plan or decomposition / product or service level. This motivates a top-down transformation of system goals that in general cannot be performed unambiguously[6] and consequently requires the use of heuristics[7].

ad i. Dynamic meta-planning can be applied, e.g. by prioritizing the decompositions respectively products or services so that the planning procedure obey the system goal as far as possible. This means it will apply the specific decomposition or use the specific product whenever possible for a given problem.

ad ii. Either static or dynamic meta-planning can be applied. Static meta-planning simply has to evaluate all plans that have been generated for the problem at hand and has to select the most profitable one. Dynamic meta-planning would have to rely on heuristics, since the system goal cannot be implemented directly at the product level. Such a heuristic could be to prefer always the most profitable product among a set of alternative ones.

ad iii. Either static or dynamic meta-planning can be applied but both have to rely on heuristics. Such a heuristic could be to prefer always the cheapest plan for the customer based on the assumption that most profit can be made if as much customers as possible can be attracted by extremely attractive offers. Static meta-planning would consequently select the cheapest plan out of the set of alternatives. Dynamic meta-planning would have to rely on a further heuristic, since the former cannot be implemented directly at the decomposition / product or service level. This second heuristic could analogously prefer always the cheapest product among a set of alternative ones.

Fig. 5 summarizes and visualizes the respective kinds of system goals.

Up to now we have (implicitly) concentrated on the highest level, the inter-enterprise level. If we want to apply the discussed concept of goal implantation on lower level mediator agents as well we need to consider the ranking/importance of the goals for these kinds of agents. For example, the mediator agent representing an enterprise needs to be highly autonomous, which means that it only acts strictly according to the objectives of its enterprise and cannot be prevented by some else to do that. This clearly indicates that goals of the agent need to be modeled as system goals. On the other hand, the agent representing an atomic system/machine has low autonomy. Only if the superordinated levels do not intervene it can react strictly according to its own goals. Therefore, here the goals of the agent need to be modeled as output goals. A more detailed discussion of intentional problem solving can be found in [38].

[6] In general it is not possible to derive the most profitable plan by simply choosing the most profitable decomposition or activity among emerging alternatives.

[7] Algorithm that can not guarantee an optimal performance in any case but an improvement for the average-case performance [[16], p. 94].

4 Conclusions

In this paper we have proposed a model for an emergent e-Logistics infrastructure for timely emergency response management by collaborative problem-solving. We have shown what kind of mediator agents are necessary in such an infrastructure and how the difficult task of flexible and dynamic goal representation on the agent level can be dealt with. The proposed e-Logistics model is applicable to a wide range of problems requiring timely configuration and coordination of distributed resources needed to address emergency situations as: *disaster emergency logistics* (evacuation planning, scheduling of emergency relief crews, food and water distribution, hospital bed planning); *national defense and security* (emergence of military holarchies as infrastructure for coordination of military operations in case of an unexpected attack); *ubiquitous ad-hoc healthcare* (emergence of a medical holarchy grouping the most suitable medical entities able to cooperate and organize their interaction to respond adequately to patient's need; medical emergency logistics with patient information retrieval and heterogeneous transaction workflow management throughout the medical holarchy); *fault-tolerant flexible production* (emergent planning and scheduling of reconfigurable manufacturing production; customer-centric supply chain and workflow management; fault tracking and error reporting across the manufacturing holarchy).

5 Future Work

In order to learn from real experiences we intend to distribute a large scale reference software platform across Canadian GAIN Nodes (to be later extended across the Global Agentcities Nodes) enabling the creation and deployment of emergency e-Logistics applications. The ultimate goal is

- to validate the emergence methodology first on a simple e-Logistics simulation test case - using and extending FIPA compliant multi-agent platforms
- to build a prototype for a medical emergency application
- to generalize the methodology into a reference model enabling quick deployment of emergency e-Logistics applications capable to integrate heterogeneous devices, resources and organizational structures into a synergetic collaborative unit synchronizing individual goals with the goals of the superordinated unit(s) at the higher level(s).

In the long term we aim to make this methodology the international standard in e-Logistics. For this we will integrate the results obtained working cooperatively with the international consortia into a *reference software platform for emergency e-Logistics*.

References

[1] □vila, Paulo, Goran D. Putnik, Maria Manuela Cunha: Brokerage Function in Agile/Virtual Enterprise Integration □A Literature Review, Proc. of the 3rd IFIP Conf. on Infrastructures for Virtual Enterprise, pp. 65-72.

[2] Cao, J. and S. Ziang: A Study on Telecollaborative Product Development, International Journal of Information Technology and Decision Making Special Issue on Virtual Organizations and e-Commerce Applications, Vol 1, No 3. September 2002, pp. 441-457.

[3] Choi, A. and H. Lutfiya: Delivering Adaptive Web Content Based on Client Computing Resources, Engineering for Human-Computer Interaction, May 2001, pp. 112-132.

[4] Crowell, Gary: Government Keynote at the Continuity from Chaos: Managing Resources and Logistics Information for Federal Enterprises, FEMA Administration and Resource Planning Directorate, Washington DC, March 19, 2002.

[5] Dennett, D. C.: *The Intentional Stance*. The MIT Press, 1987.

[6] Gerow, Bob: Emergency Management Software Solutions, Presentation at the Disaster Forum 2002 □Cooperating in Emergency Preparedness, Sept. 29-Oct 2, 2002, Calgary. AB, Canada.

[7] Greenberg, S.: Context as a Dynamic Construct, Human-Computer Interaction, Vol. 16 (2-4) p257-68, 200

[8] Holmes, Jeffrey: Industry Keynote at Continuity from Chaos: Managing Resources and Logistics Information for Federal Enterprises, Aerospace and Public Sector Manugustics Inc., Washington DC, March 19, 2002.

[9] Jennings, N. et. al.: Autonomous Agents for Business Process Management, International Journal of Applied AI, 14 (2) 2000, pp. 145-189.

[10] Luck; Michael, Vladim□ Ma řk□ Olga St ěp□nkov□ Robert Trappl (Editors): *Proc. Advanced Course on Artificial Intelligence (ACAI 2001) "Multi-Agent Systems and Their Applications (MASA)",* Prague, Czech Republic; July 2001; Joint event of ECCAI (European Coordinating Committee for Artificial Intelligence) and AgentLink (European Network of Excellence for Agent-Based Computing); Springer Lecture Notes in Artificial Intelligence (LNAI) 2086; 2002

[11] Maldonado, K. and C. Ceron: A Case Study in Emergency Relief Logistics, Logistics Quarterly (the official magazine of the Canadian Professional Logistics Institute), Summer 200

[12] McHugh, P., Wheeler, W., Merli, G.: *Beyond Business Process Reengineering: Towards the Holonic Enterprise*, John Wiley & Sons, Inc., Toronto, 1995

[13] Mentrup, C., O. Fuhrer: e-Motion: e-Mobile Testbed for Interoperability of Networks in e-Logistics,; Session on Innovative Sectorial Applications, Proc. 1st Int. Conf. on Mobile Business - Evolution Scenarios for Emerging Mobile Commerce Services, July 8-9, 2002 Athens, Greece

[14] Pechoucek, M., V. Marik, J. Barta: Knowledge Based Approach to Operations-Other-Than-War Coalition Formation, IEEE Transactions on Intelligent Systems, Special Issue on Coalition Operations, summer 2002.

[15] Perrow, Ch.: *Organizational analysis: A sociological view*. Wadsworth Pub Co; ASIN: 0818502878; (June 1970)

[16] Russell, S.; Norvig, P.: *Artificial intelligence: a modern approach*. Prentice-Hall, Upper Saddle River, NJ, 1995.

[17] Schultz, K., M. Orlowska: Towards a cross-organizational workflow model, Proc. 3rd IFIP Conf. on Infrastructures for Virtual Enterprise, May 1-3, 2002, Sesimbra, Kluwer, ISBN 1-4020-7020-9.

[18] Schwartz, J., Jr.: Incident Response Planning, (On-screen commander at the Pentagon on 9-11-01), Presentation at Continuity from Chaos: Managing Resources and Logistics Information for Federal Enterprises (A Logistics and Supply Chain Management Training Conference organized by The Association for Enterprise Integration), Washington DC, March 19, 2002.

[19] Sycara, K., S. Widoff, M. Klusch and J. Lu,: LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace, Autonomous Agents and Multi-Agent Systems, Volume 5, No. 2, June 2002, Kluwer ISSN 1387-2532.

[20] Sycara, K.: *Multi-agent Infrastructure, Agent Discovery, Middle Agents for Web Services and Interoperation;* in [10]; 2002

[21] Tolle, M., P. Bernus, J. Vesterager: Reference Models for Virtual Enterprises, Proc. of the 3rd IFIP Conf. on Infrastructures for Virtual Enterprises: Collaborative Business Environments and Virtual Enterprises, May 1-3, 2002, Sesimbra, Portugal, Kluwer 2002, ISBN 1-4020-7020-9, pp. 3-10.

[22] Ulieru, Mihaela and D. Norrie: Fault Recovery in Distributed Manufacturing Systems by Emergent Holonic Re-Configuration: A Fuzzy Multi-Agent Modeling Approach, Information Science, 7669, ISSN # 0020-0255, Sept. 2000, pp. 101-125

[23] Ulieru, Mihaela and Douglas Norrie: A Multi-Agent System for Supply Chain Management, Progress In Simulation, Modeling, Analysis and Synthesis of Modern Electrical and Electronic Devices and Systems (N. Mastorakis, Ed.), World Scientific and Engineering Society Press, ISBN 960-8052-08-4, pp. 342-348, 1999

[24] Ulieru, Mihaela and Michael Smith: A Hybrid Approach to the Automatic Design of Adaptive Control and Diagnostic Systems", Proceedings of IEEE/SMC 96 International Conference on Systems, Man and Cybernetics October 14-17,1996, Beijing, CHINA, Vol. 3, pp. 1742-1747

[25] Ulieru, Mihaela and Rainer Unland: A Multi-Agent Problem Solving (MAPS) Approach to Emergent Virtual Organizations, Submitted (October 2002) to the *International Journal of Information Technology and Decision Making*.

[26] Ulieru, Mihaela and S. Ramakhrishnan: An Approach to the Modeling of Multi-Agent Systems as Fuzzy Dynamical Systems, Advances in AI and Engineering Cybernetics, Vol. V: Multi-Agent Systems/Space-Time Logic/Neural Networks (G. Lasker, Ed.), IIAS-68-99, ISBN 0921836619, 1999,

[27] Ulieru, Mihaela and Silviu Ionita: Soft Computing Techniques for the Holonic Enterprise, FLINT 2001, M. Nikravesh and B. Azvine (Eds.), New Directions in Enhancing the Power of the Internet, UC Berkeley Electronics Research Laboratory, Memorandum No. UCB/ERL M01/28, August 200 pp 182-187.

[28] Ulieru, Mihaela, D. Norrie, R. Kremer and W. Shen: A Multi-Resolution Collaborative Architecture for web-Centric Global Manufacturing Information Sciences, Volume 127, Journal no.: 7669, ISSN # 0020-0255, Aug. 2000, pp. 3-2

[29] Ulieru, Mihaela, Dan Stefanoiu and Douglas Norrie: Holonic Self-Organization of Multi-Agent Systems by Fuzzy Modeling with Application to Intelligent Manufacturing, IEEE-SMC 2000, Nashville, USA, October, pp. 1661-1666

[30] Ulieru, Mihaela, J. Wu, M. Cobzaru, and D. H. Norrie: Agent-based Supply Chain Management Systems: State of the Art and Implementation Issues", MAMA 2000, International ICSC Congress on Intelligent Systems and Applications (ISA 2000), December 11-15, 2000, Wollongong, Australia, Vol. 1, pp. 663-669.

[31] Ulieru, Mihaela, Jianbing Wu, Mircea Cobzaru and Douglas Norrie: SC-web-CS: Supply Chain Web-Centric Systems". in Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing (ASC2000), Banff, July 24-26, 2000, pp. 501-507.

[32] Ulieru, Mihaela, M. Cobzaru and D. Norrie: □A FIPA-OS Based Multi-Agent Architecture for Global Supply-Chain Applications□, IPMM 2001 International Conference on Intelligent Processing and Manufacturing of Materials, July 29-August 3, 2001, Vancouver, BC (Proceedings on CD-Rom)

[33] Ulieru, Mihaela, Robert Brennan and Scott Walker: □The Holonic Enterprise □A Model for Internet-Enabled Global Supply Chain and Workflow Management□, International Journal of Integrated Manufacturing Systems, No 13/8, 2002, ISSN 0957-6061

[34] Ulieru, Mihaela, Scott Walker and Robert Brennan: □Holonic Enterprise as a Collaborative Information Ecosystem□, Workshop on □Holons: Autonomous and Cooperative Agents for the Industry□, Autonomous Agents 2001, Montreal, May 29, 2001, pp. 1-13.

[35] Ulieru, Mihaela: □Emergence of Holonic Enterprises from Multi-Agent Systems: A Fuzzy-Evolutionary Approach□, Invited Chapter in *Soft Computing Agents: A New Perspective on Dynamic Information Systems*, (V. Loia □Editor), IOS Press - Frontiers in AI and Applications Series 2002, ISBN 1 58603 292 5.

[36] Ulieru, Mihaela: □FIPA-Enabled Holonic Enterprise□ □Opening Address (invited) of the Workshop on Multi-Agent-Based Factory Automation, within 8th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2001), Nice, France, October 16-21, 2001, pp 453-462.

[37] Ulieru, Mihaela: □Internet-Enabled Soft Computing Holarchies for e-Health Applications□, (35 pages) in *New Directions in Enhancing the Power of the Internet*, (L.A. Zadeh and M. Nikravesh □Editors), Springer Verlag, Berlin, 2003 (in print).

[38] Ulieru, Mihaela and A. Colombo (Editors): *"Holonic Enterprises"* (M. Ulieru), Lecture Notes in Computer Science, Springer Verlag, ISBN 3-9808628-1-X Dec. 2003

[39] Ulieru, Mihaela, Dan Stefanoiu and Douglas Norrie: Holonic Metamorphic Architectures for Manufacturing: Identifying Holonic Structures in Multi-Agent Systems by Fuzzy Modeling□, Invited Chapter in *Handbook of Computational Intelligence in Design and Manufacturing* (Jun Wang & Andrew Kussiak □Editors), CRC Press 2000, ISBN No 0-8493-0592-6, pp. 3-1 □3-36

[40] Wanka, U.: *Multiagent Problem Solving*; PhD-Thesis; University of Essen; 1997

[41] Willmott, S. et. al: The Agentcities Network Architecture, Proceedings of the Workshop on Challenges in Open Agent Environments, Bologna, Italy, July 15, 2002, pp. 70-5.

[42] Wooldridge, M.; Jennings, N.: *Agent Theories, Architectures, and Languages: A Survey*. In: Wooldridge, M. and Jennings, N. (eds.): Intelligent Agents. Lecture Notes in AI 890, Springer-Verlag, 1995, pp. 1 □39

[43] Yu, B. and M. Singh: Emergence of Agent-Based Referral Networks, Proceedings of Autonomous Agents and Multi-agent Systems Joint Conf.; Bologna, Italy, July 15-19, 2002, pp. 1268-9.

An Evolutionary Approach for Studying Heterogeneous Strategies in Electronic Markets

Alexander Babanov, Wolfgang Ketter, and Maria Gini

University of Minnesota, Minneapolis, MN 55455, USA
{babanov,ketter,gini}@cs.umn.edu

Abstract. We propose an evolutionary approach for studying strategic agents that interact in electronic marketplaces. We describe how this approach can be used when agents' strategies are based on different methodologies, employing incompatible rules for collecting information and for reproduction. We present experimental results from a simulated market, where multiple service providers compete for customers using different deployment and pricing schemes. The results show that heterogeneous strategies evolve in the same market and provide useful research data.

1 Introduction

Online marketplaces are gaining popularity among producers seeking to streamline their supply chains [8], and among consumers looking for good opportunities. Intelligent software agents can significantly facilitate human decision processes either by helping to select strategies to increase profit or by making autonomous choices. In either case, agents need methods for making decisions under uncertain and changing market conditions (see [14] for an analysis of pricing strategies).

The approach to the study of agent decision making that we propose is based on a large-scale evolutionary simulation environment [4]. The goal of the simulation is to discover a set of parameters and the corresponding niche in the market where a strategy has a competitive advantage over other strategies. This knowledge, in turn, can be used to design new strategies.

The rationale behind our choice of an evolutionary framework is that it provides results without requiring an overly complex theory of agent motivation, optimization criteria, or strategic interaction. The framework is determined by the motives of individual agents, the rules of agents interactions, and the laws governing survival and creation of new agents. Given that, the evolution of the system provides dynamic information on the macroscopic behaviors of the agents society. For an explanation of the relation between micromotives of agents and macrobehavior see [26].

Evolutionary frameworks have been used extensively in Economics [20, 11, 25, 29]. One reason is that economists have long recognized the usefulness of computational systems for rigorous studies through controlled experimentation (see, for example, [17]). Evolutionary systems are relatively straightforward to construct, and, at the same time, they provide a viable tool for experimentation.

© Springer-Verlag Berlin Heidelberg 2004

Using an evolutionary approach allows one to analyze how the success of different strategies changes as the result of interactions among many agents over long periods of time.

We start by considering in Section 2 the issues and standard methods for studying the dynamics of interaction and strategies in electronic marketplaces, and we outline our proposed methodology. In Section 3 we present a case study of a simulated market, where multiple service providers compete for customers, and where profitability is the criterion used to stay in business. The experimental results we show conform to expectations. Service providers with different strategies and capacities end up occupying different niches in the market. Finally, in Section 4 we compare our approach with other related methods.

2 An Evolutionary Framework

The long-term goal of our research is to usefully employ intelligent agents to support human decision making activities in electronic markets. A theoretical analysis of sample strategies used by agents can help us understand what benefits could be derived from their use and what strategies would be most appropriate.

There are two major assumptions made frequently to enable theoretical analysis. One commonly made assumption is that the market is sufficiently large, so no individual agent has a large impact on the market. Under such assumption, the analysis can be conducted as if no other agent changes its strategy. The second common approach is to assume that everyone adopts the same strategy, and use the representative agent approach to solve for an equilibrium.

These approaches become problematic when we want to analyze a market in which many different strategies are present, and none of them clearly dominates the others. In this case, the relative strength of each strategy depends greatly on the other strategies that are present in the market as well as on the general market situation. This suggests the need for studying agent strategies in a dynamic market environment.

There is an interplay between theory and computational experiments. A good example to illustrate the issue is the case reported in [25] about his study of the labor market clearinghouse for American physicians. As reported in the study, none of the theorems from the theoretical model could be applied directly to the real market data, since the models where much simpler than reality. However, the theory proved useful to design more complex experiments and to validate the experimental results. The computational methods allowed to compare the experimental results with the theory. Showing that departures from the theory were small was an important outcome of the entire study.

We propose to use an evolutionary approach as a computational framework for assessing multi-agent systems. The major advantage of an evolutionary framework is that it is a natural choice for studying complex society of entities, where the structure of the society changes during the simulation, and the entities change as well in an effort to adapt to the changing environment. Evolutionary game theory [30] provides tools for analysis in dynamic environments. Examples studied by other researchers using evolutionary methods range from the emergence of cooperation in an otherwise selfish society [2, 3] with possible formation of spatial patterns of strategic interaction [16], to ostracism and neighborhood effects [13], to the design of auction mechanisms [24].

A drawback of most evolutionary systems is that they require a homogeneous representation of the strategies used by the agents, since the major mechanism to maintain diversity is crossbreading [18]. The homogeneity requirement, which is dictated by the agent reproduction rules, causes most evolutionary environments to be in-house projects that do not benefit from the cooperation of several research teams. In addition, even for well studied problems, it is hard to come up with a suitable representation of the strategies, as reported by [12].

In our approach, agents are reproduced not by using crossbreading but instead according to the distribution of different types of agents. Our method adds a layer of evolutionary learning atop disjointly evolving types of agents that use different strategies. Every type of agent maintains its own separate source of "genetic" information. This information could be a gene pool, if the type is based on the genetic algorithms paradigm, it could be some statistical data, as it is the case in the test model we introduce later in this paper, or it could be a neural network, which is continuously trained on the performance of its "children."

Agents who fail to satisfy a predefined performance criterion are removed at regular time intervals and, eventually, replaced by more fit entities. The purpose of the additional evolutionary layer is to learn the probabilities with which new agents of each type should enter the market. By observing the distribution of the surviving agents by type, the system can assess which types of agents are more successful and give them additional market space by creating new agents of their same type. Once the type of each new agent has been decided, the corresponding reproduction rule is applied.

We illustrate this two-layered evolutionary framework in greater detail in the next Section where we introduce our case study of a society of service providers and customers.

3 Test Model

In this Section we present a case study of how the suggested approach is used to assess two different pricing and deployment strategies in a market with multiple suppliers and their customers.

Our test model is a continuous time discrete-event simulation of a society of economic agents: suppliers of a service and their customers. The assumptions of the model are similar to those from the urban economics model of a monocentric city [22], and from the model of market areas with free entry [19]. We intentionally build our model upon classical urban economics models to ensure its practical meaning. It is important to note though that the observations of the model's behavior are not paramount, they are offered mainly to demonstrate that the results obtained by the proposed approach are, in fact, meaningful.

3.1 General Terms

The agents live and interact in a circular city of radius R. Customers come to the market for a single transaction at random intervals governed by a stationary Poisson process with a fixed frequency λ^{c}:

$$t^{\mathrm{c}}_{i+1} = t^{\mathrm{c}}_i - \frac{1}{\lambda^{\mathrm{c}}} \log U[0,1]$$

where $U[x,y]$ is a random variable distributed uniformly on the interval $[x,y]$. The location of a new customer in polar coordinates is determined by the following rules:

$$r \sim U[0,R] \qquad \text{and} \qquad \alpha \sim U[0,2\pi)$$

The rules imply that the density of customers is inversely proportional to the distance from the center of the city[1]. For convenience, we divide the city in ten concentric zones, which are equally wide so each zone gets the same number of customer entries per unit of time.

Upon entry, a customer observes different suppliers and chooses the one that provides the service at minimum cost. We define the cost of service c as a function of the supplier's price p, distance to the customer d, and delay due to servicing previously scheduled customers Δt:

$$c = p + d \times c^{\mathrm{mile}} + \Delta t \times c^{\mathrm{hour}}$$

where c^{mile} and c^{hour} are cost per mile of travel and cost per hour delay respectively.

The customer side of the society is assumed to be in equilibrium and does not change its properties in the course of a simulation. It is the society of suppliers that is expected to evolve to meet the demands of the customers and to remain in business. The restriction on the customer side is imposed to fix the scale of the simulation as well as to avoid imposing extra assumptions on the behavior of customers. In fact, it is customary for urban economics models to assume a fixed distribution of the population density and a particular form of the customers' utility function. In our model instead we have the freedom of changing the parameters of the customer side during the simulation.

Suppliers, in turn, enter the market with a frequency that is positively correlated to the average profit derived in the market. A supplier is characterized by its pricing strategy and the number of customers it can serve simultaneously, also called the supplier's *size* for brevity. Serving one customer takes size s supplier one continuous hour and costs $c^{\mathrm{work}}(s)$, staying idle for an interval of time of any length costs $c^{\mathrm{idle}}(s)$ per hour. Both costs decrease with size to simulate economies of scale.

Each supplier is audited at regular periods and removed from the market if its profit becomes negative.

[1] Studies in urban economics suggest and support by empirical evidence the use of a reversed exponential relation between population density and distance from the city center (see, for example, [1, 22]). We adopt a hyperbolic distance-density relation for convenience of the analysis and subsequent evolutionary experiments.

3.2 Supplier Strategies and Generators

We define a *supplier type* to be a pair of supplier's size and pricing strategy. A type is represented in the market by the corresponding *supplier generator*. Each generator maintains a pool of information concerning the history and the current state of its type suppliers. It uses the collected information to create new suppliers of its type and to provide them with initial location, price and, perhaps, other parameters specific to the type.

The probability that a supplier of a particular type will enter the market next is proportional to the number of suppliers of its type that are surviving in the market. There is also a small probability, referred to as *noise*, that a new supplier is assigned a type at random. The noise allows types that at some point had disappeared from the market completely to enter it again at a different time. It also suggests a way for new types to enter the market, as it will be shown later in the experimental results.

In the experiments we describe later, we have used two strategies that exhibit sufficiently different behavior. We designed the strategies so that neither strategy has a strict advantage over the other. Because of that, the strategies can coexist and evolve in the market at the same time. In both strategies, the supplier accepts whatever location and price was suggested by its generator and never alters them. Such restriction simplifies the analysis of the results, since only generators are capable of learning and adapting to the market situation.

The first strategy, code named *market sampler*, involves sampling the city in several locations to maximize a potential revenue flow given the state of the market. Under a set of assumptions this amounts to finding a maximum of $(p_c + d_c \times c^{\text{mile}})^3$, where p_c and d_c are the price and the distance to the cheapest supplier at the location. The cheapest supplier is found by sampling the market.

The second strategy, *price seeker*, assumes that the "right" price depends solely on the distance from the center of the city, not on a specific location within a zone. The price seeker generator chooses the location of a new supplier according to the distribution of the surviving suppliers of the same type. The denser the suppliers are in a zone, the more likely it is that the new supplier will be allocated to that zone. The price for the new supplier is obtained from the distribution of the prices charged by suppliers of the same type in the same zone. We assume the price distribution in each zone is normal.

To summarize the important properties of the strategies, the market sampler strategy is capable of pinpointing the best location to deploy a new supplier, yet it assumes the same price distribution for every location. The price seeker strategy can choose better prices on a global scale, but it lacks precision in positioning its suppliers relative to their competitors.

3.3 Expected Properties of the Model

There are a few properties one might expect to observe in a reasonably behaving simulation. First, large size suppliers shall eventually concentrate in the center of the city where their production cost advantage will attract customers and where the customer demand is sufficiently high to give enough work to larger suppliers. In the outer parts of the city, the scarcity of customers shall result in a low viable density of suppliers and, hence, in a relatively high influence of transportation costs on the customer preferences. That should make small suppliers prefer the rim, where they could stay closer to the customers than their larger counterparts.

The second expectation is that in a market dominated by market samplers the distribution of prices as a function of distance from the center should be flatter than in a market dominated by price seekers. The reason is that market samplers assume the same distribution of prices suits all city locations, while price seekers can crowd in zones in which their size and price are advantageous. The same argument implies that the concentration of small suppliers in the outer zones and large suppliers in the central ones shall be expected to be more distinct in a society of price seekers than in a society of market samplers.

The final expectation is related to the introduction in the market of new types of agents or of types that have disappeared from the market. In a properly functioning model it should be possible for new types of agents to acquire a niche in an existing market. This does not imply that every strategy must win a noticeable representation in the market, only that a reasonably competitive strategy should eventually get its share when the market state is favorable.

3.4 Experimental Results

In order to verify the viability of our ideas we have conducted a large number of experiments using different market parameters, such as the frequencies and costs defined previously. The following analysis applies to a typical experiment[2], which starts with a city populated by price seeker suppliers of sizes 1, 2 and 3. Later, market sampler suppliers of the same three sizes are introduced to the market via a 10% noise factor (i.e. initially each of the new types gets less than 2% chance every time a new supplier is created).

The resulting distribution of entry probabilities for each of the six types of suppliers is shown in Figure 1. Each *milestone* (m/s) in the figure corresponds to 10,000 simulated hours and roughly 2.5 million transactions[3]. The market samplers enter the market at milestone 100 when the situation is relatively stable and try to find their niche. Eventually, size 3 market sampler agents prove to be competitive and capture a sizable share of the market.

In the following we consider two simulation milestones: one at 110, soon after the market samplers' entry, and the other at 290, at the point of the major success of size 3 market sampler types. These two milestones are depicted in Figure 1 by vertical lines.

[2] For the previously introduced parameters we used the following values: $R = 50$ miles, $\lambda^{\mathrm{c}} = 250$ customers per hour, $c^{\mathrm{mile}} = 0.5$ \$/mile and $c^{\mathrm{hour}} = 1$ \$/hour. Costs c^{idle} and c^{work} for size 1 suppliers were set to 5 and 10 \$/hour respectively and reduced by 3% with each size unit gain.

[3] The probabilities are kept constant for the first 10 milestones to let the generators adjust their (or rather our) pretty volatile initial guesses about the market situation.

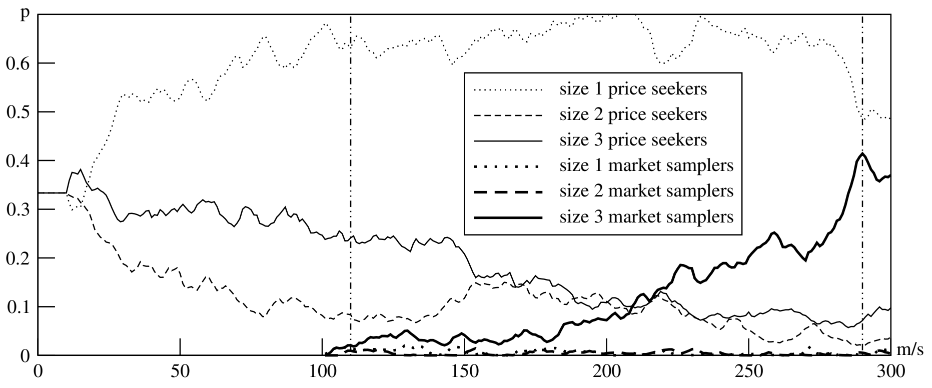


Fig. 1. Probabilities of a new supplier entry for different supplier types as a function of milestone numbers. Market sampler suppliers are introduced at milestone 100. Vertical lines denote milestones 110 and 290

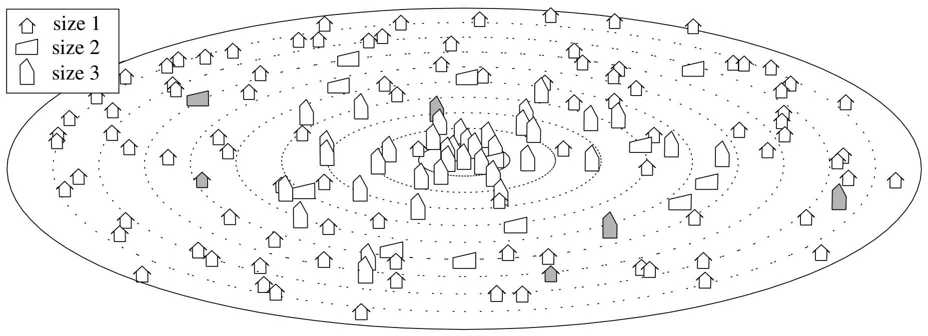


Fig. 2. City snapshot at milestone 110. Price seeker suppliers are white, market samplers are gray

Figure 2 suggests a schematic view of the city at milestone 110. Each supplier is depicted by a house of a particular shape, depending on the supplier's type; price seeker types are white and market samplers are gray. As described earlier, the concentric circles divide the city in ten equally wide zones with each zone getting roughly the same number of customer entries per unit of time.

There are two important observations to be made from Figure 2. Firstly, price seekers dominate the market and, indeed, their size 1 suppliers tend to congregate in the rim, while size 3 operate mostly in the middle of the city and size 2 form a ring in between the other two types. Secondly, the distribution of suppliers is quite uneven with dense clusters and wide open areas situated at the same distance from the center. The summary of the observations confirms that the price seeker generators have little regard to the exact placement of suppliers, while converging to the right distributions as a whole.

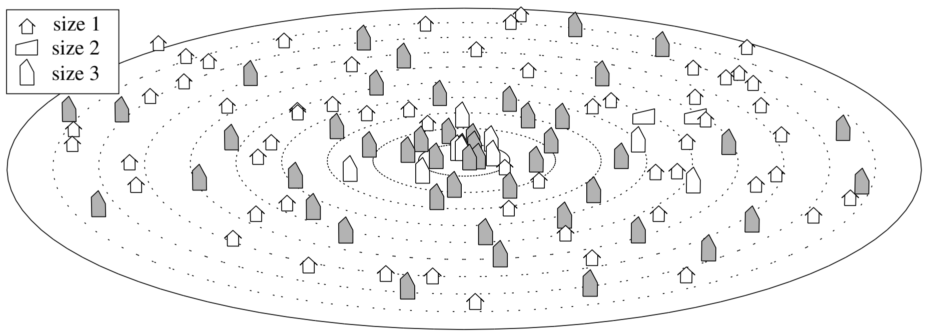


Fig. 3. City snapshot at milestone 290. Price seeker suppliers are white, market samplers are gray

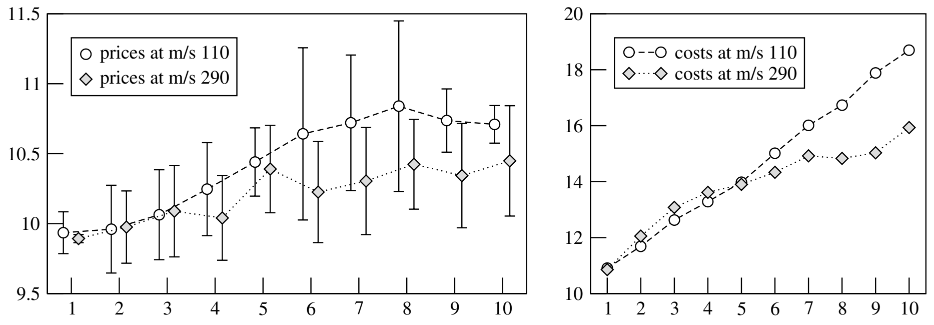


Fig. 4. Average supplier prices with standard deviations (left) and 25 hour half-life decaying averages of customer costs (right) for 10 concentric city zones at milestones 110 and 290

Figure 3 presents a view of the city at milestone 290 when the market is dominated by size 3 market samplers and size 1 price seekers. The structure of the market is evidently different from the one we witnessed 10 milestones after the market samplers were introduced — the size 3 market samplers are distributed regularly across the city with the size 1 price seekers surviving in between and in several remaining clusters.

To complete the comparison of the market situations we introduce Figure 4 with information on the prices and the costs that customers face at the two considered milestones. The left graph shows the average prices and standard deviations for the ten concentric city zones. The right graph shows the *decaying averages* for the customer costs in the same zones. The concept of the decaying average is introduced to avoid storage requirements for calculating moving averages in a large scale discrete event simulation. We define the decaying average da_i with a half-life T for the time interval $[t_i, t_{i+1})$ iteratively as

$$w_i = 1 + w_{i-1} 2^{-\frac{t_i - t_{i-1}}{T}} \quad \text{and} \quad da_i = \frac{v_i + (w_i - 1) da_{i-1}}{w_i}$$

where t_i and v_i are respectively the value and the time of the event i, and w_i is the weight of all events from 1 to i at time t_i. Both graphs suggest that the market populated with market sampler types exhibits more even price and, subsequently, more even customer cost distributions.

It should be emphasized here that the most important conclusion of our experiments is that a society of agents that use strategies based on different approaches, information pools, and reproduction methods does evolve as one society and produces meaningful results.

4 Related Work

Much research has been done in the last few years in designing pricing strategies for agents, assessing their performance, and seeing how well they adapt to changing environmental situations [5].

Understanding collective interactions among agents that dynamically price services or goods is discussed in [14], where several pricing strategies are compared. The major difference between their pricing strategies and the strategies our agents use is that transportation costs and geographical locations are important for us, but are not relevant when buying and selling information in an electronic form. They describe examples of price-wars caused by agents that dynamically change their price for information bundles. Their results show how the pricing decisions that an agent makes affect the market and, consequently, the decisions of other agents. Because of the complexity of the problem, the experiments they present are limited to a small number of agents.

A simulation based approach to study dynamic pricing strategies in finite time horizon markets is described in [10]. The study uses a market simulator and simple strategies. The results are evaluated in terms of overall profit, but there are so many variables in the simulation that it is hard to assess the generality of the results obtained. One of the advantages of using an evolutionary framework is that it simplifies experimenting by internalizing a variety of parameters that otherwise would be necessary to describe relationships between agents.

An important conclusion of many studies is that even simple strategies can be very effective. For instance, Cliff's [6] Zero-Intelligence Plus trader agents have minimal intelligence, yet they have been successfully used in continuous double auctions, where they performed very well even when compared to human traders [9].

The use of evolutionary methods is proposed in [23], who simulates the evolution of the agent population as they adapt their strategy for continuous double auctions by observing what happens in the environment. Cliff [7] uses genetic algorithms to learn the parameters that control how his trader agents evolve their pricing strategies. Along similar lines, an evolutionary system based on Genetic Programming is presented in [24]. In this system, agents evolve auction strategies to bid in the electricity market.

The major difference between these and the work presented here, is that we are interested in understanding how strategies of individual agents interact in the market, as opposed to study specific types of auctions to learn auction rules.

The evolutionary game theory suggests a variety of stability concepts: from a static and rather unrestrictive Evolutionary Stable Strategy [28] to a strong Evolutionary Robustness [21] in dynamic games. An Evolutionary Stable Strategy has to satisfy two conditions: (i) it can survive in isolation, and (ii) no mutation adopted by an arbitrarily small fraction of individuals can invade it by getting at least the same payoff. In our system a strategy only needs to satisfy the first requirement. The second is almost never true, since invading strategies are often able to secure a sizeable portion of the market, yet not able to take over the market completely. One might think of an appropriate stability concept as of “competitive” or “niche” stability.

We are interested in providing a methodology for effectively studying multi-agent systems with a large number of agents. Studying such systems analytically is often impossible. There are a few attempts to model very large multi-agent systems at the macroscopic level. Shehory [27] models them using ideas from classical mechanics. Goal satisfaction is modeled by collisions between dynamic particles, the agents, and static particles, the goals. The method requires a measure of distance to the goal, which is hard to do except in cases where agents operate in a Cartesian environment. The methodology presented in [15] is limited to systems that obey the Markov property, i.e. such that the agent’s future state depends only on its present state.

5 Conclusions

We have proposed an evolutionary framework where agents with different pricing and deployment strategies compete in the same market. New agents are introduced to the market with a probability proportional to the number of agents of the same type already in the market. New strategies and strategies that have disappeared from the market may be introduced at any point.

In our current framework, only generators are capable of learning and adapting to the market situation. Once a supplier has been created, it will stay in its location and it will keep the price it charges to customers constant for the entire simulation, or until it is removed from the market for poor performance. This makes the choices made by the generators specially important for the survival of each agent type.

We have presented results using two different strategies to select the suppliers location and price. During the simulations we observed how different types of suppliers occupied different market niches and how new behavior patterns emerged over time. The referred test case illustrates that in such evolutionary system several *optimal* strategies, not only a single *optimum* one, can survive in the market after it stabilizes.

Our current work is aimed at developing an analytical model of the society of agents, and analyzing how close the experimental results obtained with the evolutionary framework are to the results predicted by the model.

Acknowledgements

Partial support for this research is gratefully acknowledged from the National Science Foundation under awards NSF/IIS-0084202 and NSF/EIA-9986042.

References

[1] Alex Anas, Richard Arnott, and Kenneth A. Small. Urban spatial structure. *Journal of Economic Literature*, 36(3):1426–1464, September 1998. 160

[2] R. M. Axelrod. *The evolution of cooperation*. Basic Books, 1984. 159

[3] Robert Axelrod. *The complexity of cooperation*. Princeton University Press, 1997. 159

[4] A. Babanov, W. Ketter, and M. Gini. An evolutionary framework for large-scale experimentation in multi-agent systems. In T. Wagner, editor, *MAS Problem Spaces and Their Implications to Achieving Globally Coherent Behavior*. Kluwer, 2003. 157

[5] Christopher H. Brooks, Robert Gazzale, Rajarshi Das, Jeffrey O. Kephart, Jeffrey K. MacKie-Mason, and Edmund H. Durfee. Model selection in an information economy: Choosing what to learn. *Computational Intelligence*, April 2002. 165

[6] D. Cliff and J. Bruten. Minimalintelligence agents for bargaining behaviors in marketbased environments. Technical Report HPL-97-91, Hewlett Packard Labs, 1997. 165

[7] Dave Cliff. Evolutionary optimization of parameter sets for adaptive software-agent traders in continuous double auction markets. Technical Report HPL-2001-99, Hewlett Packard Labs, 2001. 165

[8] John Collins, Wolfgang Ketter, and Maria Gini. A multi-agent negotiation testbed for contracting tasks with temporal and precedence constraints. *Int'l Journal of Electronic Commerce*, 7(1):35–57, 2002. 157

[9] Rajarshi Das, James E. Hanson, Jeffrey O. Kephart, and Gerald Tesauro. Agent-human interactions in the continuous double auction. In *Proc. of the 17th Joint Conf. on Artificial Intelligence*, Seattle, WA, USA, August 2001. 165

[10] Joan Morris DiMicco, Amy Greenwald, and Pattie Maes. Dynamic pricing strategies under a finite time horizon. In *Proc. of ACM Conf on Electronic Commerce (EC'01)*, October 2001. 165

[11] Joshua M. Epstein and Robert Axtell. *Growing Artificial Societies: Social Science from the Bottom Up*. The MIT Press, Cambridge, MA, 1996. 157

[12] Stephanie Forrest. Genetic algorithms: Principles of natural selection applied to computation. *Science*, 261:872–878, 1993. 159

[13] Richard J. Gaylord and Louis J. D'Andrea. *Simulating Society*. Springler-Verlag, 1998. 159

[14] Jeffrey O. Kephart, James E. Hanson, and Amy R. Greenwald. Dynamic pricing by software agents. *Computer Networks*, 32(6):731–752, 2000. 157, 165

[15] Kristina Lerman. Design and mathematical analysis of agent-based systems. In *Lecture Notes in Artificial Intelligence (LNAI 1871)*, pages 222–233. Springer Verlag, 2001. 166

[16] Kristian Lindgren. Evolutionary dynamics in game-theoretic models. In *The Economy as an Evolving Complex System II*, pages 337–367, 1997. 159

[17] D. McFadzean, D. Stewart, and Leigh Tesfatsion. A computational laboratory for evolutionary trade networks. *IEEE Transactions on Evolutionary Computation*, 5(5):546–560, October 2001. 157

[18] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996. 159

[19] Edwin S. Mills and Michael R. Lav. A model of market areas with free entry. *Journal of Political Economy*, 72(3):278–288, 1964. 159

[20] Richard R. Nelson. Recent evolutionary theorizing about economic change. *Journal of Economic Literature*, 33(1):48–90, March 1995. 157

[21] Jörg Oechssler and Frank Riedel. On the dynamic foundation of evolutionary stability in continuous models. *Journal of Economic Theory*, 107:223–252, 2002. 166

[22] Arthur O'Sullivan. *Urban Economics*. McGraw-Hill Higher Education, 1999. 159, 160

[23] Sunju Park, Edmund H. Durfee, and William P. Birmingham. An adaptive agent bidding strategy based on stochastic modeling. In *Proc. of the Third Int'l Conf. on Autonomous Agents*, 1999. 165

[24] Steve Phelps, Peter McBurney, Simon Parsons, and Elizabeth Sklar. Co-evolutionary mechanism design: a preliminary report. In J. Padget, Onn Shehory, David Parkes, Norman Sadeh, and William Walsh, editors, *Agent Mediated Electronic Commerce IV*, volume LNAI2531, pages 123–142. Springer-Verlag, 2002. 159, 165

[25] Alvin E. Roth. The economist as engineer: Game theory, experimentation, and computation as tools for design economics. *Econometrica*, 70(4):1341–1378, July 2002. 157, 158

[26] Thomas C. Schelling. *Micromotives and Macrobehavior*. W. W. Norton & Company, Inc., 1978. 157

[27] Onn Shehory, Sarit Kraus, and O. Yadgar. Emergent cooperative goal-satisfaction in large scale automated-agent systems. *Artificial Intelligence*, 110(1), May 1999. 166

[28] J. Maynard Smith. *Evolution and the Theory of Games*. Cambridge University Press, Cambridge, December 1982. 166

[29] Leigh Tesfatsion. Agent-based computational economics: Growing economies from the bottom up. *Artificial Life*, 8(1):55–82, 2002. 157

[30] Jörgen W. Weibull. *Evolutionary Game Theory*. The MIT Press, 1995. 158

Self-Organizing Agents for Mechanical Design

Davy Capera, Marie-Pierre Gleizes, and Pierre Glize

IRIT □Universit□Paul Sabatier
118, Route de Narbonne, 31062 Toulouse, Cedex 4, France
`+33 561 558 343`
`{capera,gleizes,glize}@irit.fr`
`http://www.irit.fr/SMAC`

Abstract. This paper describes an automated process for designing mechanical systems based on the adaptive multi-agent system theory. At the beginning of the design process, the designer adds the elements of his problem: goal, envelope, constraints, known mechanical components □ During the solving process, mechanical components (previously □agentified□) organize themselves in order to find a solution, by modifying their parameters, by creating new mechanical components fitting in with the needs, or by modifying the system topology. While this paper presents an overview of the theoretical basis of AMAS theory, it primarily focuses on the method for developing the Mechanical Synthesis Solver and the results from the first prototype.

1 Problematics

Mechanical design consists in assembling mechanical components such as links (bars, rigid bodies) and joints (hinges, cams□), in order to build a system which performs an objective function like following a precise trajectory. The kinematic structure contains the essential information about which link is connected to which other links and by what types of joint. Moreover, designer may wish to satisfy additional constraints on available types of components: maximal number of components, mechanism weight, maximal dimensions, bounding envelope□

In the SYNAMEC[1] project (SYNthesis of Aeonautical MEChanisms), mechanical design is composed of two phases: preliminary design and detailed design. The preliminary design process -which is a pure kinematics study- is composed of three sub-phases:

1. Search for the topology of the mechanism (□type synthesis □), in other words establish how many rigid bodies are used in the system and in which way they are linked. At this level, the type of joints between bodies and their dimensions are not considered. Thus, criteria are related to more □abstract□considerations such as degrees of freedom of the mechanism and its dimensional space (planar mechanism or 3D one).

[1] SYNAMEC is a Growth European project involving SAMTECH (Belgium), ALA (Italy), CRANFIELD University (England), INTEC (Argentina), PAUL SABATIER University (France), SABCA (Belgium), SNECMA (France).

© Springer-Verlag Berlin Heidelberg 2004

2. Search for the best alternative from the topology, in other words instantiate the linkages: for instance, a planar mechanism can only contain prismatic or roto�dal joints.
3. Dimensional synthesis. In this phase, components dimensions must be adjusted to fit in with the given goal of the system (following a trajectory for example).

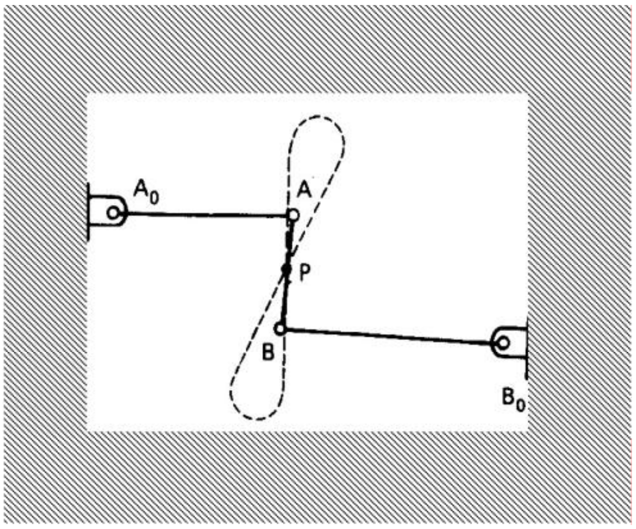


Fig. 1. Example of mechanism

Fig. 1. shows a mechanism example the designer has to provide. The mechanism, which is made up by tree bars and four hinges (A0, A , B, B0), follows the dotted trajectory at the point P. Mechanism is contained in a bounding envelop on which the attachment points are (A0 and B0).

A mechanism is created by the engineer helped by his intuition, ingenuity and experience. He processes usually by trials and errors by using some computer aided design tools such as Catia™ from Dassault Systemes[2] or Samcef[3] Field Mecano™. This process could be very time consuming and synthesis techniques are not very used in industry. After the designer completes the preliminary design, the resulting mechanism should be modified and even discarded at the sight of problems which occur during the next design phases. Thus, design is the creation of synthesized solutions in the form of products or systems that satisfy customer�s requirements. Design is a continuous process of refining customer requirements into a final product. The process is iterative in nature and solutions are usually not unique. It involves a process of decision making. Recent methodological approaches based on systematic creation and classification of mechanisms to arrive at the best mechanism in shorter time are proposed by Tsai [18].

In order to solve this mechanical design problem, we propose a very different approach based on the Adaptive Multi-Agent Systems theory (AMAS theory). The

[2] http://www.3ds.com/en/brands/catia_ipf.asp
[3] http://www.samcef.com/

principle is to solve the problem by a self-assembling process in which the mechanical components organize themselves from their interactions and the feedback of the environment. Mechanical components (rigid bodies and joints) are agentified and the mechanism builds itself by self-organization of its agentified components. The originality of this approach resides in the manner the re-organization rules work and in the fact that they are completely independent from the function the system has to achieve.

So, an automated tool for mechanical design task would be able to autonomously find mechanisms for the given goals and constraints. Nevertheless, this is a complex task because the combinatorial explosion between the basic mechanical components involves a huge space search. This complexity is increased when two other points are taken into consideration:

1. The designer cannot know the number and type of components the problem requires.
2. The topological modification of a mechanism leads to change drastically its global function: the solution space search is discontinuous and no relevant heuristic exists.

The second section of this paper presents briefly the AMAS theory which is the foundation of our way to handle adaptive systems. In the second part, we apply this theory to mechanical synthesis by explaining the overall process and the behaviour of the basic agentified components. The fourth section gives some results of this approach for XBars problems. We analyze these results and compare them with related works in the section five.

2 AMAS Theory

In the AMAS theory (6]), a multi-agent system is a system composed of autonomous entities interacting in a common environment. A Multi Agent System (MAS) has also an environment and it can reach a behavioral or a functional adequacy. For example, in a simulation, reaching a behavioral adequacy is to reproduce the behavior of the simulated entity; a functional adequacy is to perform the right task, the task for which the system had been built. We are specifically interested in Adaptive MAS. Classically such a system is defined by the fact that it is a MAS which is able to change its behavior to react to the evolution of its environment and has the following characteristics:

- the system is evolving in an environment with which it interacts,
- the system is composed of different parts: the agents,
- each agent has its own function to compute,
- the global function of the system is not implemented in the agents and there is no global control,
- the design is a bottom-up one: agents are firstly defined.

In our vision, the important notion is the collective; the AMAS theory must then lead to a coherent collective activity that realizes the right task. This property is called "functional adequacy" and the following theorem has been proved 13:

"For any functionally adequate system, there is at least a cooperative interior medium system which fulfills an equivalent function in the same environment".

Therefore, we focused on the design of cooperative interior medium systems in which agents are in cooperative interactions.

The specificity of our AMAS theory resides in the fact that we do not code the global function of the system within an agent. Due to the agents' self-organization ability, the system is able to adapt itself and realizes a function that it is not coded in the agent, that is emerging and this is due in part to the interactions between components. If the organization between the agents changes, the function which is realized by the collective changes. Each agent possesses the ability of self-organization i.e. the capacity to locally rearrange its interactions with others depending on the individual task it has to solve. Changing the interactions between agents can indeed lead to a change at the global level and this induces the modification of the global function. This capacity of self-organization at the lowest level enables to change the global function without coding this modification at the upper level of the system and so the system adapts itself.

Self-organization is founded on the capacity an agent possesses to be locally "cooperative", this does not mean that it is always helping the other ones or that it is altruistic but only that it is able to recognize cooperation failures called "Non Cooperative Situations" (NCS, which could be related to exceptions in classical programs) and to treat them. The local treatment of NCS is a means to build a system that does the best it can when a difficulty is encountered. Such a difficulty is primarily due to the dynamical nature of the environment of the system, as well as the dynamics of the interactions between agents. More precisely an agent can detect three kinds of NCS:

1. when a signal perceived from its environment is not understood and not read without ambiguity;
2. when the information perceived does not induce the agent to an activity process;
3. when concluding results lead to act in a useless way in the environment.

An AMAS-compliant system is emergent because its global function is not coded within the agents. Like for natural systems, the adaptation occurs when the system is functioning in a given environment. At the conception phase, designers have mainly to fill in a nutshell associated to each generic class of agents with its cooperative behavior as described above.

3 Mechanical Synthesis Solver

The development of the software to generate mechanisms has begun with the University of CRANFIELD for mechanical advices and SAMTECH for the kinematic simulation engine called MECANO. In this section, we present the global functioning of the system and the mechanical agents specification.

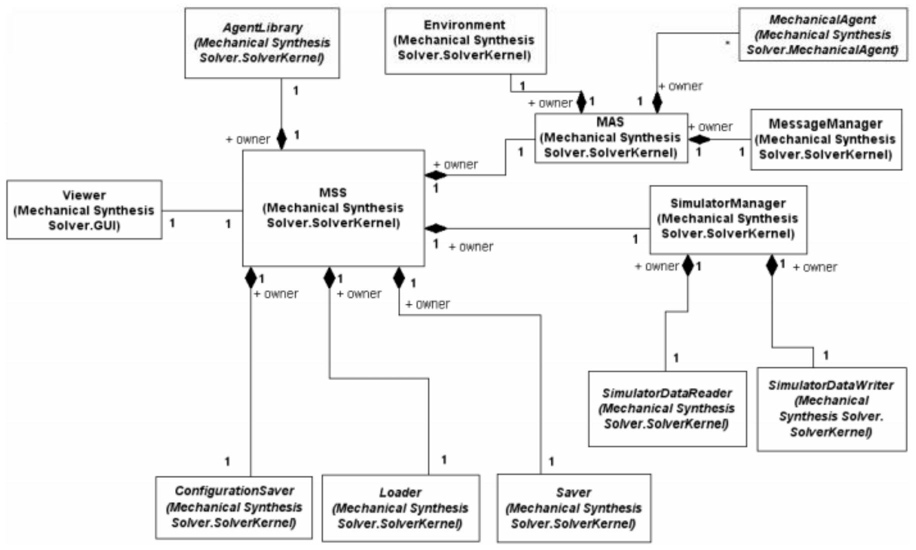

AgentLibrary (Mechanical Synthesis Solver.SolverKernel)
Environment (Mechanical Synthesis Solver.SolverKernel)
MechanicalAgent (Mechanical Synthesis Solver.MechanicalAgent)
MAS (Mechanical Synthesis Solver.SolverKernel)
MessageManager (Mechanical Synthesis Solver.SolverKernel)
Viewer (Mechanical Synthesis Solver.GUI)
MSS (Mechanical Synthesis Solver.SolverKernel)
SimulatorManager (Mechanical Synthesis Solver.SolverKernel)
SimulatorDataReader (Mechanical Synthesis Solver.SolverKernel)
SimulatorDataWriter (Mechanical Synthesis Solver. SolverKernel)
ConfigurationSaver (Mechanical Synthesis Solver.SolverKernel)
Loader (Mechanical Synthesis Solver.SolverKernel)
Saver (Mechanical Synthesis Solver.SolverKernel)
+ owner
1
*

Fig. 2. General class diagram of the *Mechanical Synthesis Solver*

3.1 The Process

The mechanism moves in a specific environment, generally defined by a trajectory, an envelope and a movement control. Thus, the multi-agent technique needs a simulation engine which is able to compute mechanical movements (interactions between components, collision with envelope) and to return information about the state of each component (position, forces and loads, for example) to the multi-agent system. During the analysis phase of this problem, an agreement between partners has been reached to use MECANO as the environment engine. Thus, the software learns a relevant mechanism by using a loop composed of the cycle: MECANO simulation followed by the agents self-organization (Figure 4). This is the basic learning evolutionary process of the mechanism. Each cycle is decomposed into three phases:

1. Simulation engine computes the motion of the current mechanism.
2. Data related to the new mechanism state are sent to the mechanical agent in order to update them.
3. The AMAS tool performs □optimization□ □resolving non cooperative situations detected by the mechanical agents compliantly the AMAS theory- that leads to a new mechanism.

The interface with the simulator is performed through files which describe, in input, characteristics of the mechanism, the environment and the commands related to the whished motion (the □actuation function□), and allow to recover the new mechanism characteristics after the motion performed by MECANO.

Figure 2 shows the global class diagram of the Mechanical Synthesis Solver specified with UML (Unified Modeling Language) notation.

Design of the multi-agent system was performed in three steps (see 3.2 for details):

1. Identification of agents: mechanical components are represented by agents.
2. Specification of non cooperative situations and related actions: this work was performed in collaboration with the CRANFIELD University.
3. Implementation of agents from a generic model.

These development steps are compliant with the methodology of AMAS-based systems design called ADELFE (French acronym for Atelier pour le Developpement de Logiciels Fonctionalit Emergente) which is currently under development ([3]).

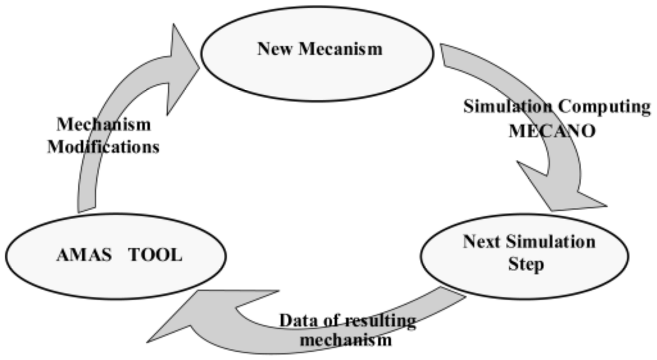


Fig. 3. Adaptation process for type synthesis of mechanisms

3.2 The Mechanical Agents

Following the ADELFE methodology, the design of *Mechanical Synthesis Solver* has focused on two main steps: the identification of agents and the specification of the non cooperative situations and related actions for each type of agent.

3.2.1 Identification of Agents

Every basic mechanical component of the component library (chosen from the MECANO element library) will be an autonomous entity which has to collectively self-organize. Each autonomous agent is able to manage its own mechanical-related characteristics (for instance, length for a bar) and has some agent-related capabilities (communication, memorization, believes management). The prototype which is currently developed includes the following components as agents: hinges, bars, attachment points and trajectory points.

A trajectory point agent is an exception: it does not represent any mechanical component. Its goal is to fit with the trajectory. It can perceive the distance between its position and the goal trajectory and interprets any situation for which this distance is different from zero as a non cooperative situation. A trajectory point agent can fix itself on rigid bodies and define its own attachment position on the rigid body.

3.2.2 Non Cooperative Situations and Related Actions

As explained in section 2, a designer of an AMAS-based system has to exhaustively specify the non cooperative situations which may occur for every agent. A mechanical expert advice is necessary for the definition of these non cooperative situations which lead the self-organizing process. They are situations in which an agent decides to change its internal parameters (adjustments) and/or its place in the current mechanism (topology transformation). The detection of theses situations has to be only performed on the local available information of an agent.

This work was performed in collaboration with mechanical experts from CRANFIELD University ([7]). All situations are derived from the cooperation definition given in section 2, and may be classified in three types: incomprehension, incompetence and uselessness ([13]).

As agents share the same application field and a common language, thus ambiguity or misunderstood situations cannot occur (see case one in the AMAS theory part). The following table (Table 1) is an example of a uselessness situation for a bar agent (case three in the AMAS theory):

Table 1. Uselessness situation for bar a agent

Name	Uselessness
Description	The agent is totally useless in the mechanism.
Condition(s)	The agent is linked with none other agent.
Action(s)	Find a link with a join agent Create a new agent join agent Suicide

Obviously, when a mechanical component is separated from the global mechanism it is totally useless. In this case, its corresponding agent tries to integrate it or, when failing, it disappears. This situation is defined in the same way for other agents but actions □links□and □creations□can only concern □rigid body agents□(such as bar agents).

Some situations are related to the goal and can be linked with incompetence situations. Here is an example of such a situation for a bar agent:

Table 2. Goal incompetence situation for a bar agent

Name	Goal incompetence
Description	Direct perception of a stiffness related to a difference between ideal and real trajectory
Condition(s)	The value of □goal stiffness□is not nil
Action(s)	Change position of the bar extremities Break a link with one of its adjacent agents

In this case, the agent perceives local information (the stiffness see 5.1) indicating that the environment is not satisfied by the behaviour of the system. At this stage, the agent must decide if this feedback is related to its own activity by taking into account his beliefs and memory about previous interactions. This feedback allows also the agent to adjust its beliefs about the relevance of its actions and the other known agents.

Observing the corpus of agent actions, one can see that self-organizing approach for mechanical design allows discovering topologies from scratch: at the beginning, the mechanism can be empty. The self-organization process is composed of four phases:

1. Component apparition. When the current topology is sub-optimal (for example there exists some residual stiffness for the trajectory point), a new component is added from the set of generic agents.
2. Component reorganization. When a given component of the current alternative is frequently in uncooperative situation, it tries to modify its location (for example a bar links with another component of the system).
3. Component adjustment. When uncooperative situation occurs in a component, it tries to self-adjust in a better way (for example a bar can modify its length). Thus, the global system finds the better response for a given alternative.
4. Component disappearance. When a component had a great amount of uncooperative situations, he thinks to be useless. The component deletion occurs only after trying to find a relevant location inside the mechanism.

Thus, the core process of self-organization is deciding if the current alternative is relevant to achieve the goal of the mechanism. In this paper, results of this core work from two usual examples in mechanical design, the XBars problem, are described.

4 Results

The first developed version is a prototype with reduced functionalities: it only focuses on the basic X-bar problems solving. These problems consist in finding mechanisms whose one point must follow a given trajectory. These mechanisms are composed of basic components such as bars, hinges and attachment points (to fix the mechanism on the envelope). So, the agents that are available in the prototype are related to these three mechanical components. They are also linked to an agent trajectory point which represents the point that must follow the trajectory defined by the user.

For this first prototype, we only test the dimensions adjustment problem which takes place with the third sub phase of the preliminary design: the mechanism has to adapt the length of the bars in order to the point fits with the given trajectory.

4.1 3-Bars Problem

The 3-Bar problem consists in a mechanism composed by three bars, four hinges and two attachment points. The goal trajectory is a vertical straight line.

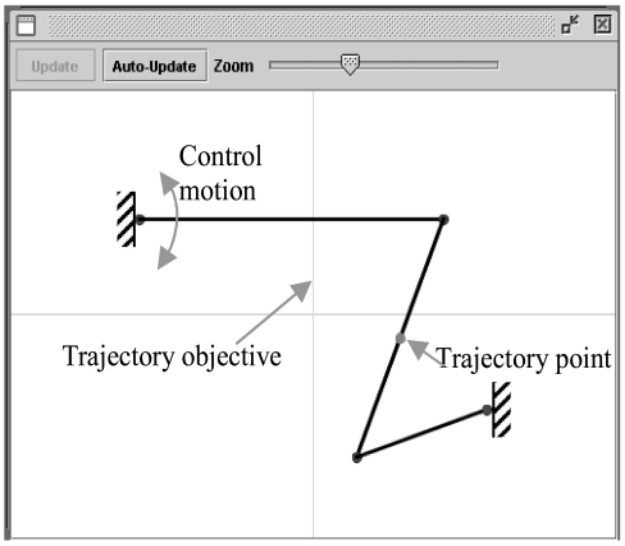


Fig. 4. Initial state of a bad-dimensioned 3-Bar mechanism

The Figure 4 shows an initial state with bad dimensions- of a 3-Bar mechanism (the centered straight line is the trajectory, and the trajectory point is on the middle of the central bar):

The actuation is a rotation on the upper left hinge: it performs cyclically a wipe of 0.4 radians by steps of 0.05 radians. A cycle is the number of steps motion the system realizes in order to find the same position. In this case a cycle is composed of 16 steps

In figure 5, the curve measures at each step the error (abscissa) of the mechanism, i.e. the distance between the point and the trajectory (ordinate). This figure shows that the mechanism adjusts itself quickly (less than one cycle) and reaches a configuration in which the distance is lower than 0.1.

In this simulation, the process runs 48 steps that correspond to 4 cycles. After the great changes obtained during the first steps, the adaptation very slightly improves the global behavior.

4.2 5-Bars Problem

The 5-Bar problem consists of a mechanism composed of five bars, four hinges and two attachment points. The goal trajectory is a horizontal straight line.

The picture below shows an initial state with bad dimensions- of a 5-Bar mechanism (the straight line at the bottom of the figure represents the trajectory and the trajectory point is at the bottom of the triangular structure):

The actuation is a rotation on the upper left hinge: it cyclically performs a wipe of 0.4 radians by steps of 0.05 radians. Here, there also are 16 steps for a cycle.

In the figure 7, the abscissa and ordinate have the same meaning as in the figure 5. The figure shows that the mechanism converges toward a solution and that the distance between the trajectory and the point decreases. However, solving is slower

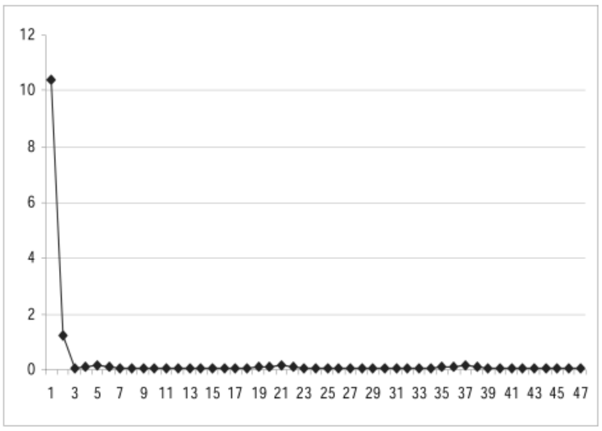


Fig. 5. Distance to the trajectory (ordinate) of a 3-Bar mechanism as the function of step number (abscissa)

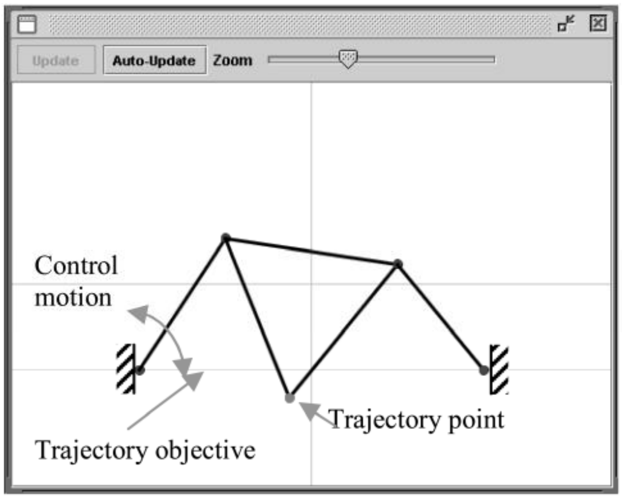


Fig. 6. Initial state of a bad-dimensioned 3-Bar mechanism

than for the 3-Bars, because the maximum distance during the 40th cycle is still about 0.12. This comes from three reasons:

1. A 5-Bars mechanism contains more components than a 3-Bars one. This gives more fuzzy information to local agents and leads to adjustments that are not always relevant.
2. Stiffness (see 5.1) comes from geometry (and not cinematic) analysis of the mechanism. Thus, the derived agent adaptation is sometimes relatively poor.
3. All cooperative behaviors are not yet implemented inside agents. This certainly reduces the adaptation efficiency for the more complex mechanisms.

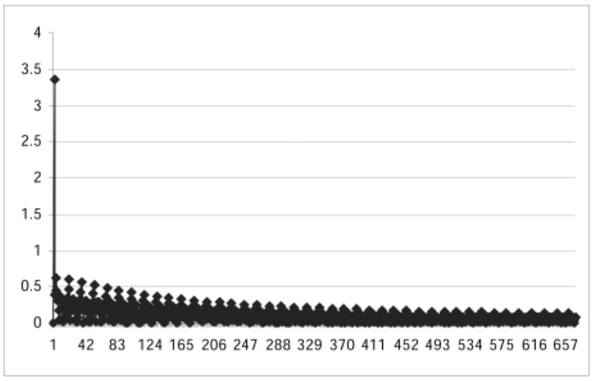


Fig. 7. Distance to the trajectory (ordinate) of a 5-Bar mechanism as the function of step number (abscissa)

4.3 Further Developments

The results are satisfying and the further objectives are foreseen as:

- Implementation of the full self-organizing process for agents: adding non cooperative situations and related actions to process topological modifications and to allow adding/deletion of agents. The specification has already been performed with experts from CRANFIELD University ([7]).
- Definition of new types of agent to complete the set of available agents: the specification has already been performed with experts from CRANFIELD University (7]).

The adjustments-related actions will be improved in order to obtain a quicker convergence time. This improvement will be at the agent behavior level and more precisely in the action module. Indeed, the current data processing that calculates the modifications is not optimal and can be improved: for instance, by adding a memory of the previous actions an agent performed in order to take them into account.

5 Analysis and Comparison

The results given by the MSS for the 3-Bars and the 5-Bars problems are similar to those computed directly (without learning) by MECANO when we give the theoretical dimensions. This indicates the relevance and efficiency of the local adaptation process derived from the AMAS theory.

5.1 Emergence

In the section 2, we claim that systems design following AMAS techniques provides solutions that are emergent. Actually, the mechanical agents behavior does not contain any part which directly processes the distance to trajectory decrease. Fig. 8.

shows correlation between the local non cooperation measure of agents and the distance to the trajectory. The top dots represent the average distance to the trajectory during a complete cycle (16 steps) and the lower ones represent the average non cooperation degree or stiffness- of the whole system (the sum of non cooperation degrees of all the agents).

The stiffness depends on the distance error between the trajectory point and the trajectory objective. This global information is the feedback of the environment (returned by the MECANO tool) but unknown by the agents. Nevertheless the geometric approach computed by MECANO allows giving a local stiffness associated to each agent. This local stiffness is neither exactly informative to the magnitude of adjustment to apply nor its direction. This information is derived from the cooperative learning process of the agents.

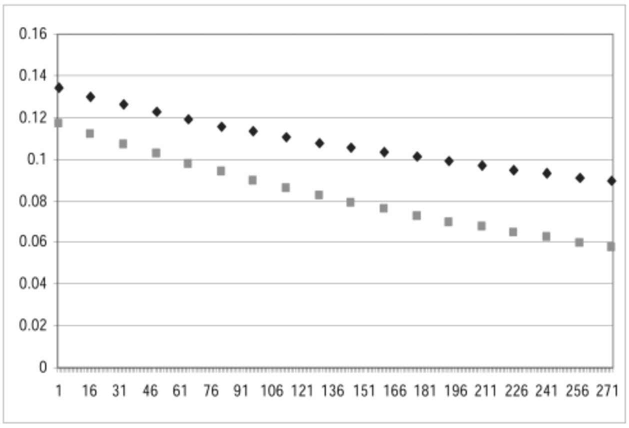


Fig. 8. Distance to the trajectory (up) and stiffness

There is a strong correlation between the local and global criteria of non cooperation (distance to the trajectory is considered as the non cooperation measure of the whole system). That experimentally shows that the local reasoning of the parts of a system leads to a global emergent behavior.

5.2 Related Works in Mechanical Field

There is very few works on preliminary design problematics in literature; besides the A-Design application which is based on a global evolutionary algorithm and a mechanism synthesis method develop by Lung-Weng Tsai, only tools which support designers exist (DAO, CAO, simulators).

5.2.1 A-Design

This tool was developed by Campbell, Cagan and Kotovsky ([4]) from Carnegie Mellon University. The solving process is a classical evolutionary one whose selection, generation and reproduction operations are performed by multi-expert systems based on agent principle (one agent for each expert module) and blackboard architecture. Authors argue in 4]: *This system differs from previous work in that 1) designs are not only generated but iteratively improved upon to meet objectives specified by a designer, 2) synthesis of designs is performed within a rich descriptive representation of components and configurations that models real-world component interactions, and 3) key design alternatives are retained to allow the system the flexibility to adjust to changes in the problem description made by the designer throughout the design process.*

However, this method is very time and memory consuming because the system has to handle a set of mechanisms whereas In the Mechanical Synthesis Solver, we only deal with one mechanism which adapts. Moreover, modification and evaluation operations are complex to develop and are related to the problem the system has to solve while mechanical agents of the Mechanical Synthesis Solver always have the same behavior which is related to their function and to the non cooperative situations independently of the problem. As the solving process is based on the global evaluation of the mechanism, this method clashes with the need to deal with a discontinuous space search. AMAS theory provides a solution to develop systems which are able to avoid this clash because the global solution is not known by the agents and emerges.

5.2.2 Mechanism Synthesis Method

Recently a new approach based on an abstract representation of the kinematic structure has evolved, it is extensively developed in the book of Lung-Weng Tsai *The kinematic structure contains the essential information about which link is connected to which other links by what type of joint. It can be conveniently represented by a graph and the graph can be enumerated systematically using combinatorial analysis and computer algorithms* 18]. This methodology is composed of three main steps:

1. Enumeration of all possible solutions (according some functional requirements) using graph theory and combinatorial analysis.
2. Creation of a class of feasible mechanisms based to the remaining functional requirements.
3. Iteration on candidates in order to find the most promising for the product design.
4. This methodology has been successfully applied in the structure synthesis of many domains. In the Synamec project, Alberto Cardona ([8]) is working on a tool based this methodology. This software will use genetic algorithm to evaluate candidate mechanisms.

As in the MSS tool, this approach based on kinematic structure needs to evaluate a given mechanism at a time. Nevertheless the evaluation engine on MSS is exactly the same we use to transform the current mechanism in order to find a more relevant, and theoretically all the functional requirements could be considered for this evaluation. We think that these considerations increase the search efficiency and thus reduce the time to conception.

5.3 Works Leaning on Self-Organization Using Cooperation

Hogg and Huberman in [16] showed that when agents cooperate in a distributed search problem, they can solve it faster than any agent working in isolation. A similar result was obtained by Mataric in 19] with a set of mobile robots foraging in order to bring back tiles to "home". She has observed that when the number of individualist robots increases, the global performance decreases due to the interfering activities. For her, the ideal result will be obtained with robots having altruistic behaviors.

Steels in 23] have taken an interest in self-organization in the case of two applications. The first concerns the foraging of a geographical zone by several robots. The self-organizing mechanism is similar to the one made by ants when they try to find food a depositing of crumbs simulates the pheromone depositing, which guides other robots. The second application concerns the autonomous elaboration of a vocabulary by several agents present in a given environment. Each agent possesses for that its own set of associations word/meaning. So it can experiment associations used by other agents present in the system according to the following 3 manners: either by propagating the association it possesses, or by creating a new association or by self-organization (based on retroaction mechanism and allowing the agent to bring up to date the confidence it has in its association).

Cooperation was extensively studied in computer science by Axelrod [2] and Huberman 17] for instance. From the initial studies about the condition of cooperation emergence in societies, the question was now extended to the more general problem of the emergence of norms (see Flentge, Polani and Uthmann 11] on this question). "Everybody will agree that co-operation is in general advantageous for the group of co-operators as a whole, even though it may curb some individual□s freedom." (Heylighen 15]). Relevant bio-inspired approaches using cooperation are the swarm algorithms (Kennedy and Eberhart [26]) and the ants algorithms (Dorigo and Di Caro [9]) which give efficient results in many domains. We have shown in 24] that natural ants have a behaviour very closed (but not perfectly) to the cooperative one we gave in the AMAS theory.

Multi-agent learning relies on, or even requires, the presence of multiple agents and their interactions. Many authors in this domain (Goldman [14], Sekaran 21], Sen 22], Wei□ [24]) have studied the role of social behavior of agents on the global performance. They found that cooperation between agents improves the results. If we consider each agent of the system as a piece of knowledge, these works mean that knowledge is well learnt when it is organized in a cooperative manner. This is a criterion independent of the meaning (the semantic needed for common knowledge), and thus could be a good approach for a general learning theory based on cooperation.

6 Conclusion

This paper presents a new approach to solve autonomously the preliminary synthesis problem. The main principle of Mechanical Synthesis Solver -based on the Adaptive Multi-Agent System theory- is to □agentify□mechanical components and gives them a cooperative behavior: they are able to locally detect non cooperative situations and to solve them. We also present results from a prototype of this software which lead us to claim that this approach is relevant.

The AMAS theory and its application to mechanical design is a mix between two main approaches:

1. Works on problem solving based on multi-agent systems where agents have a great autonomy in order to adapt their behaviour according to their local situations. This idea is closed to ants algorithms [9] or particle swarm optimization [26] in adding explicit agent cooperation reasoning to guarantee a coherent space search process.
2. Works directed by fitness decision like genetic algorithms (see A-Design in □5.2.1) and derived algorithms based on evolution of population of □solutions□ and using a fitness function (particle swarm optimization algorithms [26] and other evolutionary algorithms using implicit or explicit operators). Nevertheless our fitness criterion is not directly based on global function evaluation, but a local one for each agent and its cooperative behaviour with its neighbourhood).

Further development will focus on the improvement and completion of agent behaviors in order to perform autonomously the type synthesis phase of design.

In order to test a theory (by falsification or validation) a great amount of experiments must be done in various fields. For this reason, we have implemented our theory on cooperative self-organization on many applications:

1. The first was the tileworld game (Piquemal 20]) in which we have experimentally verified that cooperative agents have better results than selfish ones.
2. The second concerns an application of cooperative information systems with France-Telecom (Camps 5]) and with Deutsch Telekom (Athanassiou 1]). In this software, the agents representing the users and the services create a dynamic network of mutual interest based on the cooperative self-organization process.
3. The third is about a national multi-disciplinary project about natural and artificial collective intelligence. The results of a cooperative self-organized ants society application gives performances at least better than natural simulated insects (Topin [24]).
4. The fourth is a real-time application for flood forecast (Georg□ 12]). This software runs in the organism in charge of crisis situations in the Midi-Pyr□n□es region of France and depending of the Ministry of the environment.

Usual search algorithms (stochastic or determinist), which impose the knowledge of a cost function have the same global efficiency when the corpus of examples are sufficiently large. □*In our investigation of the search problem from this match-f-to-a perspective, the first question we addressed was whether it may be that some algorithm A performs better than B, on average. Our answer to this question, given by the NFL theorem is that this is impossible*□10]. Surprisingly, we have observed (using the same algorithm pattern derived from AMAS theory) the same performances in all the applications cited previously. An explanation of this fact (not a demonstration) could be the inexistence of global cost functions for theories allowing the emergence of the global function: the global function is unknown and obviously we cannot have an associated cost function. This could be a reason to investigate massively the field on really emergent algorithms. This reflexion is based on experiences where:

1. We can know, as an observer, what is the global optimum. For example in the X-bar problems, there are optimal known solutions and we can judge the result given by the system.
2. Though the search space has a huge amount of local minima, the system avoids them. We have observed this in applications such as flood forecast, ant foraging, time tabling□

References

[1] Athanassiou E., L□ger A., Gleizes M.P., Glize P. - Abrose : Adaptive Brokerage Based on Self-Organisation Services and Users □Short paper on □European Conference on Modelling Autonomous Agents in Multi-Agent World □- 1999.

[2] Axelrod R. 1984. The Evolution of Cooperation, Basic Books, New York.

[3] Bernon C., Gleizes M.P., Peyruqueou S., Picard G. □ADELFE, a Methodology for Adaptive Multi-Agent Systems Engineering □In Third International Workshop "Engineering Societies in the Agents World" (ESAW-2002), 16-17 September 2002, Madrid.

[4] Campbell, Cagan & Kotovsky □Agent-based Synthesis of electro-mechanical design configurations □In Proceedings of DETC98 1998 ASME Design Engineering Technical Conferences September 13-16, 1998, Atlanta, GA.

[5] Camps V., Gleizes M.P. - Cooperative and mobile agents to find relevant information in a distributed resources network, Workshop on Artificial Intelligence-based tools to help W3 users, Fifth international conference on World Wide Web □1996.

[6] Capera D., Georg□J.P., Gleizes M.P., Glize P. - Emergence of organisations, emergence of functions □AISB 2003.

[7] Capera D.: Integrated Expert Advisor tool □SYNAMEC deliverable, January 2003.

[8] Cardona A. □Computational Methods for Synthesis of Mechanisms □10-02-2002.

[9] Dorigo M. and Di Caro G.. The Ant Colony Optimization Meta-Heuristic □In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization - McGraw-Hill, 1999.

[10] David H. Wolpert and William G. Macready - No free lunch theorems for search - Tech. Rep. SFI-TR-95-02-010, Santa Fe Institute □1995.

[11] Flentge F., Polani D., Uthmann T. - On the Emergence of Possession Norms in Agent Societies - Journal of Artificial Societies and Social Simulation vol. 4, no. 4 http://www.soc.surrey.ac.uk/JASSS/4/4/3.html - 2001.

[12] Georg□J.P., Gleizes M.P., Pierre Glize, R□gis C. □Real-time Simulation for Flood Forecast: an Adaptive Multi-Agent System STAFF □In Proceedings of the AISB□03 symposium on Adaptive Agents and Multi-Agent Systems, University of Wales - 2003.

[13] Gleizes M.P., Camps V., Glize P. □A Theory of emergent computation based on cooperative self-organization for adaptive artificial systems □In Fourth European Congress of Systems Science, Valencia, 1999.

[14] Goldman C.V., Rosenschein J.S. 1994. Emergent Coordination through the Use of Cooperative State-Changing Rule □AAAI.

[15] Heylighen f. (1992): Evolution, Selfishness and Cooperation; Selfish Memes and the Evolution of Cooperation, Journal of Ideas, Vol 2, # 4, pp 70-84.

[16] Hogg T., Huberman B.A. 1992. Better than the best: The power of cooperation - Lectures notes in complex systems - Addison Wesley.

[17] Huberman B.A. 1991. The performance of cooperative processes - In emergent computation, Edited by Stephanie Forrest - Special issue of Physica D.

[18] Lung-Weng Tsai: Mechanism design: Enumeraion of kinematic structures according to function. CRC Press, ISBN: 0-8493-0901-8.

[19] Mataric Maja J. 1994. Interaction and Intelligent Behavior PHD of Philosophy Massachussetts Institute of Technology May 1994.

[20] Piquemal-Baluard C., Camps V., Gleizes M.P., Glize P. - Cooperative agents to improve adaptivity of multi-agent systems, Intelligent Agent Workshop of the British Computer Society, In Specialist Interest Group on Expert Systems & Representation and Reasoning 1995.

[21] Sekaran M., Sen S. 1995. To help or not to help - Seventeenth Annual Cognitive Sciences Conference - Pitsburg Pennsylvannia.

[22] Sen S., Sekaran M. 1995 Using reciprocity to adapt to others IJCAI.

[23] Steels L. 1996 . The spontaneous Self-organization of an Adaptive Language, Machine Intelligence 15 - Oxford University Press - Oxford - Muggleton S. (Ed.) http://arti.vub.ac.be/www/steels/mi15.ps - 1996.

[24] Topin X., Fourcassi V., Gleizes M.P., R gis C., Th raulaz G., Glize P. - Theories and experiments on emergent behaviour : From natural to artificial systems and back ECCS 1999.

[25] Wei G. 1993. Learning To Coordinate Actions In Multi-Agent Systems - in Proceedings of the International Joint Conference on Artificial Intelligence.

[26] Kennedy, J. and Eberhart, R. - Particle Swarm Optimization -IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, 1995.

Can Tags Build Working Systems? From MABS to ESOA

David Hales and Bruce Edmonds

Centre for Policy Modelling
Manchester Metropolitan University
Aytoun Building, Manchester M1 3GH, UK
dave@davidhales.com
http://www.davidhales.com
be@bruce.edmonds.name
http://bruce.edmonds.name

Abstract. We outline new results coming from multi-agent based social simulation (MABS) that harness "tag-based" mechanisms in the production of self-organized behavior in simulation models. These tag mechanisms have, hitherto, been given sociological or biological interpretations. Here we attempt to move towards the application of these mechanisms to issues in self-organizing software engineering – i.e. how to program software agents that can self-organize to solve real problems. We also speculate how tags might inform the design of an unblockable P2P adaptive protocol layer.

1 Introduction

It would seem that one of the major goals that inspired the agent-based approach (at least initially) was the idea of systems that could self-organize functionality to solve problems [12, 20]. Agents would work together, dynamically employing each other's skills to achieve their individual and collective goals [13]. The key here was that a human engineer would not be needed to pre-specify the exact nature of this kind of future interaction (i.e. it would be self-organized).

Although a lot of effort has been directed at the infrastructural base required to realize this goal (languages, architectures, protocols, standards and platforms) much less effort seems to have been directed at the more fundamental issues – i.e. how to make it all "hang-together" [12, 13]. More specifically, if we view an agent based system as a kind of artificial society composed of various types of artificial agents then what kinds of mechanisms do we need to implement to allow those agents to form productive teams or institutions? How do we get agents to self-organize their society, rather than requiring the MAS engineer to program everything directly (limiting the complexly possible with current techniques)? Put another way, can we generate and apply a kind of "artificial social science" to the engineering of MAS? What might this kind of "science" look like and how would it be applied?

Over the last ten years a broad multi-disciplinary approach has formed which aims (at least in part) to address this very issue [3]. Often termed "Artificial Societies" [4, 2] when applied to possible or abstracted systems and Agent Based Social

© Springer-Verlag Berlin Heidelberg 2004

Simulation (ABSS) when applied to human systems, a growing body of knowledge is being generated. For an overview of current work see contributions to the Journal of Artificial Societies and Social Simulation (JASSS) . Social simulation has always been involved with "messy" systems [24] (i.e. human systems) and consequently has explored mechanisms that may explain order within them.

Within the broad area of Artificial Societies, Multi-Agent Based Simulation (MABS) and Agent Based Social Simulation (ABSS) a number of socially inspired models of self-organized behavior have been explored. In this paper we outline some recently presented "Tag Based Models" (TBMs) [5, 6, 7, 8, 18] and indicate how, we believe, the mechanisms identified in these models may begin to be applied to the construction of self-organizing software.

1.1 Tag Based Models

The use of "tags" for the facilitation of self-organization in artificial systems was first explicitly discussed by Holland [11]. Tags are identifiable markings or cues attached to agents that can be observed by other agents. Tags can also evolve or change in the same way that behaviors can evolve and change (for adaptive agents). In a system in which agents evolve artificial genotypes that determine their behavioral traits, tags can be viewed as phenotypically visible portions of the genotype. Early TBM's demonstrated the powerful cooperative effect that tags may foster within some simple repeated interaction scenarios [17].

More recently, TBMs have been advanced that show cooperation and altruism evolving in one-time interactions [5, 6, 18, 22] and facilitating modest forms of agent specialisation and group formation [7, 8]. Also some TBMs demonstrate "reverse scaling" properties where the introduction of more agents produces efficient solutions more quickly [8].

1.2 Paper Outline

Here we outline initial ideas towards the application of the mechanisms identified in these models to the engineering of working information systems that have the ability to self-organize. In another paper [8] we described a simulation model in which we applied a TBM to a simulated robot warehouse-unloading scenario first described by [15, 16]. There we demonstrate that robots controlled by a tag-based mechanism outperformed so called "socially rational" [14] and "selfish" behavior. Firstly we will outline the original scenario and results obtained and then we will described how these mechanisms might be applied in a dynamic peer-to-peer environment to self-organize load-balancing strategies.

2 The Warehouse Scenario

Robots must work together to service the arrival of deliveries to a warehouse. There are ten loading bays into which trucks can arrive to be unloaded at any time (if the bay is empty). To each bay five robots are assigned to unload the trucks. While working, the robots unload at a constant rate. Robots are rewarded according to how much of the goods in their bay have been unloaded. Robots may help in the unloading of other bays but receive no reward for this. All the bays start empty. If a bay is empty, a truck of size s may arrive with a probability of p in each cycle. The truck stays there until it is empty, then it leaves. Thus the probability, p, is inversely related to the time the bay is empty and the truck size, s, related to the contiguous unloading time.

Each robot has an integer tag T [1..500] which is visible to all the other agents. The only function of the tag is that it is visible to others – it has no other direct significance. The tag-based mechanism is an evolutionary process that acts upon the tags (as well as other behavioral traits) of the robots as they do the unloading. In each cycle each agent can do a fixed number of units (5) of unloading. For each unit, if the agent has a lorry in its home bay it asks another agent for help. First it sees if there is another robot with an identical tag and asks them, if there is not it asks a randomly selected robot. Whether the asked robot responds with help depends upon its strategy, which is composed of two Boolean values: whether to help if it already has its own lorry to unload (L); and whether to help if it does not have a lorry to unload (N) – i.e. if it is idle. Thus the asked robot consults one of L or N depending on whether it has a lorry in its bay and acts accordingly.

At the beginning of each cycle the T, N and L value triples are reproduced into the robot population probabilistically in proportion to the amount that the robot who had them had unloaded in its own bay. With a certain probability of mutation (0.1) these values are replaced with a random new value. Thus successful tags and response strategies are preferentially replicated to the robots for the new cycle. It is therefore likely that if a robot is relatively successful it will be replicated more than once and there will then be at least one other robot with the same tag to cooperate with. No contracts or joint / group utilities are used – the cooperation that emerges is entirely due to the implicit emergence of groups of robots with the same tags. It is a kind of solution to a commons tragedy without central planning [9].

The above outlined tag-based agent decision making is compared to the two 'hard-wired' reference cases, where robots are all "selfish" or "social". In the selfish case robots never help another robot and, hence, only unload trucks in their own bay. In the social case all robots unload from their own bay if there is a lorry there and always help another if there is not and it is requested to do so (i.e. robots will help others if idle and requested to do so). Each simulation was run for 500 cycles (which allows for each robot to unload a total of 2500 units). Statistics were collected from the runs including the percentage of time the robots were idle.

Figure 1 shows for each kind of decision function (or strategy) the total amount of time units wasted by robots sitting idle. An efficient system will minimize this value. Results are given for three different loading scenarios (different values of p and s). When p is small but s is large then deliveries are more sporadic, with lots of help required in bursts, but when p is high and s is low then jobs are arriving at a smother constant rate.

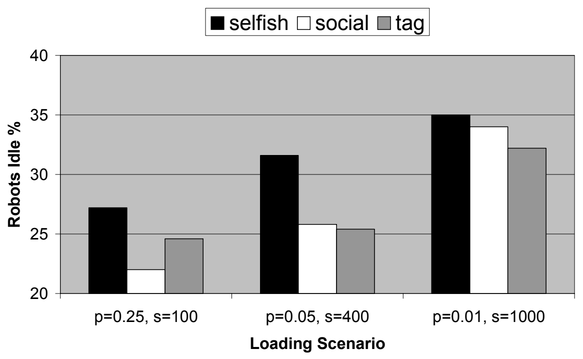


Fig. 1. Total robot idle time for each loading scenario as a percentage of total robot time

The results given in figure 1 demonstrate a very interesting property of the tag strategy. As expected the selfish strategy performs poorly since idle robots wont help others in other bays, the social strategy performs more efficiently than the tag strategy when p is high and s is low. However, for the two other loading scenarios the tag strategy outperforms the hardwired social strategy. It would seem that the system has self-organized to respond to the sporadic loading scenarios – we are currently investigating the exact dynamics that lead to this very encouraging result. We speculate however, that the tag strategy allows agents to abandon unloading their own trucks in order to help with a newly arriving truck – which would potentially reduce idleness. This is something that the hardwired social agents cannot do. Here we believe we have found an example of the tag system producing a better solution than an intuitively hardwired socially rational solution. Indeed the original purpose of applying tags to the model was to explore an alternative to conventional modes of agent rationality [19, 21].

3 Self-organized P2P Using Tags

We now outline how the simulation above, which involved robots in a warehouse, might be applied to a peer-to-peer environment in which client/servers need to distribute jobs.

Assume we have a networked environment in which a number of peer client/servers (which we will call agents) handle job requests passed to them from users. The jobs might be information request requiring the searching and filtering of various data sources or even processing jobs requiring the application of computational resources. In any case, each agent can only handle a certain number of jobs at a given time. An agent may, if it is already servicing several jobs, ask another agent to deal with a request. However, the process of locating another agent and making the request would entail computational costs. Let us impose an additional constraint in this hypothetical scenario – assume that credit (however determined) is apportioned from the user to the original agent requested (if the job is performed adequately) no matter which agent actually tackled the job. This latter point simplifies the situation somewhat; since we only require users to connect to a single agent and issue a job request, if the request is met then credit is apportioned in some manner (this could be as simple as some kind of direct payment, or less direct such as causing increasing use of the agent).

We argue here that if we recast our robots from the warehouse as agents in our network and arriving lorries as job requests from users then we have the basis for a self-organizing load balancing system. How would "reproduction" of a new generation based on credit be achieved in a network peer-to-peer environment? The only condition for this would be that agents periodically compared their credit against that being attained by another and copy the strategies and tags of the other (if the other was attaining higher credit). Such a mechanism would be enough to drive an evolutionarily process. We argue that the simulation results from the warehouse scenario are applicable here – although the minor differences (i.e. each agent receiving direct credit and a slightly different evolutionary mechanism) ultimately require testing in a new simulation model.

Such a tag-based method of job distribution in a dynamic environment would incur minimal overheads – since no central planning, or large amount of information passing is required.

3.1 Specialization and Cooperation

Here we describe a further TBM [7, 8] and in a following section again suggest how the model may be applied to a self-organizing software application scenario. In this model agents are placed in an environment where the formation of groups with internal specialization would produce a beneficial collective effect. Agents are selected at random and awarded resources. However, to "harvest" the resources agents require the correct skill. If they do not have the correct skill they may pass the resource to another agent with that skill (at a cost to themselves) or simply discard the resource. Each agent stores a single integer number representing its skill {1, 2, 3, 4, 5} (an agent can only store one skill). Each agent stores two real values: a tag from [0..1] and a tolerance (also in [0..1]). An agent considers another to be in its "group" if the absolute difference in their tag values is less-than-or-equal to the tolerance value of the agent. Agents with higher utility (derived from harvested resources) are periodically copied by others and mutation is applied with some small probability to skill, tag and tolerance values.

During each cycle agents are selected at random and given some number of resources. Each resource is associated with a skill (randomly selected). If the agent posses the correct skill then it will harvest the resource (and gain one unit of utility) but if the agent does not posses the required skill then it will search the population for an agent that is part of it's group with the required skill and pass (donate) the resource (at a cost to itself). It would seem that agents with low tolerances (and hence small or empty groups) would perform well because they would not incur the costs of donation – that is, they would act selfishly. However, a collectively rational solution would be for groups to form with a diversity of skills, that way each member of the group would benefit by passing appropriate resources among each other. After all agents have had some number of awards (this value was varied see below) agents are reproduced probabilistically based on their relative utility to form a new population (the population size was fixed at 100 for this model).

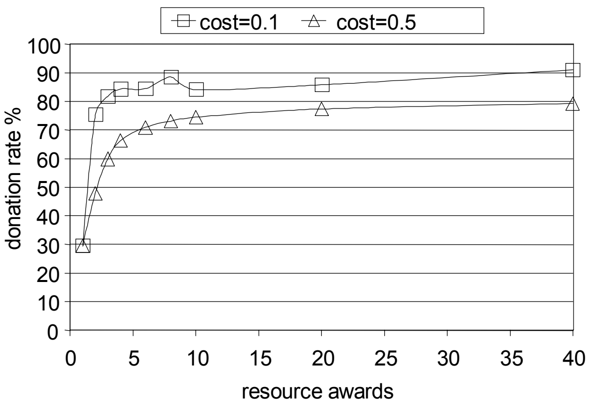


Fig. 2. Donation rates against number of resources awarded to each agent in each cycle

Figure 2 shows results from simulation of the model. Results are averages over 5 runs each to 30,000 cycles. The number of resource awards is the number of times each agent is given a resource (in each cycle of the simulation) that it can keep, donate or discard. The donation rate is the proportion of times agents donate the resources they are awarded. It indicates the percentage of attempted donations (passes of resources) that result in success (i.e. find a recipient). Notice that the results show that neither pure individual rationality nor collective rationality dominates completely for all numbers of resource reward. However, remember that for high donation rates to occur agents must form groups that contain a diversity of skills. Consequently we are asking a lot more from a simple learning process. Given this we still note that even when the cost of donation is half that of the benefit (cost = 0.5) to the receiving agent, there is still a donation rate of 30% for the lowest number of resources awarded per agent per cycle (this being 1 award). This is far from what individual rationality (individual optimization) would produce (0%) but still means that 70% of resources are being wasted by the system – far from full collective rationality. However when the number of resources awarded to each agent is increased this value quickly increases. When a resource award is set to 5 the donation rate is above 70% for both cost levels. This result indicates that a sufficient number of resources need to be awarded to agents before they adapt their behaviors. Considering the simplicity of the learning system and the requirement that agents have to produce groups that are internally specialized this is a very promising result.

3.2 Emergent Specialization in P2P

Consider a P2P environment similar to that described previously. In that scenario each agent (client/server node) could process any job that was passed to it by a user. In this sense each agent was a generalist – it could potentially accept any job. The agents could ask for help from others if they were overloaded with jobs. Consider a more realistic kind of scenario in which each agent can deal with some subset of possible jobs, for example some agents may be able to locate information on the web, others may be able to store and retrieve information at a later date others may offer certain kinds of processing (for example, converting a document from one format to another).

We argue the results from the previous simulation indicate that tags may be used to address this problem. A user can pass any agent any job; the agent will then either accept the job (if it has the correct abilities) or pass the job (if it does not have the correct skills) or discard the job. The above simulation results indicate that even if the act of locating another agent with the required skills is costly to the referring agent, this may still evolve, from a process of copying those who are doing relatively better. Again a relatively simple process involving tags (i.e. no central control, contracts or sophisticated redistribution of credit) can produce efficient levels of self-organized structure.

Obviously such a model is very simplistic, more sophisticated agents could learn when to apply their skills to a job (learning of a context or domain of application) [1] and also adapt their behavior to improve their job performance.

3.3 An Unblockable Adaptive P2P Protocol?

We speculate that the cooperation producing process outlined above (for detail see [5]) in PD might be interpreted as the dynamics over time of a set of P2P file sharing systems. Each unique tag represents a unique protocol – i.e. tags are protocols. Agents sharing protocols can interact – cooperation involves productive sharing of computational and data resources (file sharing) and defection involves some non-cooperative activity (not sharing, hogging bandwidth, spyware deployment). Mutation of the tag represents an adaptation or innovation of a protocol. Mutation of a strategy represents innovation of action or even external blocking mechanisms imposed by network providers.

Assuming that agents could intelligently adapt protocols (implemented on-top of the IP and, say, producing permutations of HTTP user / client interaction masquerades such that packet content contained no simple identifiable pattern over time) and communicate those new protocols to their in-group (i.e. those sharing the current protocol) then the tag results obtained in the simulations encourage us to conjecture that this could be a basis for an unblockable adaptive P2P protocol since protocol innovation would be endogenous and fast (assuming thousands or millions of peers). At present this is merely a conjecture and requires further development.

4 Conclusion

We do not claim to have delivered a blue-print for self-organizing software applications in this brief paper, rather we wish to demonstrate the at least some ideas coming from the ABSS community have some possible future application in self-organizing information systems. We appreciate that we are far from this goal and that, indeed, we require parallel developments in infrastructural technologies. We only offer an initial step in what could be a productive direction.

However, we argue that via rigorous empirical analysis of ABS and the movement of algorithms from sociological or biological interpretations to information technology solutions and the construction of a series of simulation models – moving closer to implementable systems – some open issues in harnessing self-organization may be productively addressed.

In the first model we showed there that there is no need for complex credit allocations or contracts etc, that tags were sufficient to self-organize collective behavior that (in some task environments) would outperform what would seem to be the intuitively optimal (collective rational) solution (i.e. help when idle, ask when busy). In the second model we demonstrated that a simple form of internal specialization could be attainted using tag processes.

We are encouraged by our initial forays, which seem to suggest that algorithms and techniques embedded in models created for one purpose may be applied for different purposes. Our ambitious hope is that future work may continue to identify, reproduce and test various emergence-generating algorithms producing a kind of typology of emergence techniques and their scope of application.

Acknowledgements

The warehouse model was programmed in SDML [23]. Other models were programmed in JAVA2 (www.sun.com). For conversations on the topic of unblockable adaptive P2P protocols thanks go to Jeremy Goslin (www.jeremygoslin.com). We also wish to thank Scott Moss for creating the CPM (http://cfpm.org) and his commitment to the simulation approach.

References

[1] Edmonds, B. (2002) Learning and Exploiting Context in Agents. Proceedings of the 1s International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Bologna, Italy, ACM Press, 1231-1238. July 2002

[2] Epstein, J. and Axtell, R. Growing Artificial Societies: Social Science from the Bottom Up. Cambridge MA: The MIT Press, 1996.

[3] Gilbert, N. and Doran J., (eds.) Simulating Societies: the Computer Simulation of Social Phenomena. London: UCL Press, 1994.

[4] Gilbert, N. and Conte, R. (eds.) Artificial Societies: the Computer Simulation of Social Life, London: UCL Press, 1995.

[5] Hales, D. Cooperation without Space or Memory – Tags, Groups and the Prisoner's Dilemma. Multi-Agent-Based Simulation (eds. Moss, S., Davidsson, P.), Lecture Notes in Artificial Intelligence 1979. Berlin: Springer-Verlag, 2000.

[6] Hales, D. Tag Based Co-operation in Artificial Societies. Ph.D. Thesis, Department of Computer Science, University of Essex, UK, 2001.

[7] Hales, D. Evolving Specialisation, Altruism and Group-Level Optimisation Using Tags. Multi-Agent-Based Simulation II (eds. Sichman J., Davidsson, P.), Lecture Notes in Artificial Intelligence 2581. Berlin: Springer-Verlag, 2003.

[8] Hales, D. & Edmonds, B. Evolving Social Rationality for MAS using "Tags". Proceedings of the AAMAS 2003 Conference Melbourne. ACM Press.

[9] Hardin, G. The tragedy of the commons. Science, 162:1243-1248, 1968.

[10] Hogg, L. M., and Jennings, N. R. Socially Rational Agents. Proc. AAAI Fall symposium on Socially Intelligent Agents, Boston, Mass., November 8-10, 61-63, 1997.
[11] Holland, J. The Effect of Labels (Tags) on Social Interactions. SFI Working Paper 93-10-064. Santa Fe Institute, Santa Fe, NM. 1993.
[12] Jennings, N. R. Agent-based Computing: Promise and Perils. Proc. 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99), Stockholm, Sweden. (Computers and Thought award invited paper) 1429-1436. 1999.
[13] Jennings, N. R. On Agent-Based Software Engineering. Artificial Intelligence, 117 (2) 277-296, 2000.
[14] Jennings, N., and Campos, J. Towards a Social Level Characterization of Socially Responsible Agents. IEE Proceedings on Software Engineering. 144(1):11-25, 1997.
[15] Kalenka, S. Modelling Social Interaction Attitudes in Multi-Agent Systems. Ph.D Thesis. Department of Electronics and Computer Science, Southampton University, 2001.
[16] Kalenka, S., and Jennings, N.R. Socially Responsible Decision Making by Autonomous Agents. Cognition, Agency and Rationality (eds. Korta, K., Sosa, E., Arrazola, X.) Kluwer 135-149, 1999.
[17] Riolo, R. The Effects of Tag-Mediated Selection of Partners in Evolving Populations Playing the Iterated Prisoner's Dilemma. SFI Working Paper 73-02-016. Santa Fe Institute, Santa Fe, NM. 1997.
[18] Riolo, R., Cohen, M. D. and Axelrod, R. Cooperation without Reciprocity. Nature 414, 441-443, 2001.
[19] Russell, S. Rationality and Intelligence. Artificial Intelligence, 94(1):55-57, 1997.
[20] Russell, S. and Norvig, P. Artificial Intelligence: A Modern Approach. Prentice-Hall, 1995.
[21] Russell, S. and Wefald, E. Do The Right Thing: Studies in Rationality. MIT Press, 1991.
[22] Sigmund & Nowak. Tides of tolerance. Nature 414, 403-405, 2001.
[23] Moss, S., Gaylard, H., Wallis, S. and Edmonds, B. SDML: A Multi-Agent Language for Organizational Modelling. Computational and Mathematical Organization Theory, 4, 43-69. 1998
[24] Moss, S. Policy analysis from first principles. Proceedings of the U.S. National Academies of Science, Vol. 99, Supp. 3, pp. 7267-7274, 2002.

Self-Organizing Referral Networks: A Process View of Trust and Authority*

Pınar Yolum[1] and Munindar P. Singh[2]

[1] Department of Artificial Intelligence, Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
pyolum@few.vu.nl

[2] Department of Computer Science, North Carolina State University
Raleigh, NC 27695-7535, USA
singh@ncsu.edu

Abstract. We are developing a decentralized approach to trust based on referral systems, where agents adaptively give referrals to one another to find other trustworthy agents. Interestingly, referral systems provide us with a useful and intuitive model of how links may be generated: a referral corresponds to a customized link generated on demand by one agent for another. This gives us a basis for studying the processes underlying trust and authority, especially as they affect the structure of the evolving social network of agents. We explore key relationships between the policies and representations of the individual agents on the one hand and the aggregate structure of their social network on the other.

1 Introduction

This paper considers two problems, a practical and a theoretical, and comes up with an interesting interrelationship between them. The practical challenge is how to engender trust in decentralized settings consisting of autonomous parties. In such settings, we cannot rely on a centrally trusted authority. In fact, current approaches based on certificate authorities are fundamentally impoverished because they do not take into account actual evidence about the behavior of a given party, i.e., whether it has been observed to be trustworthy or not by the parties that have interacted with it. Reputation mechanisms, such as the one on eBay, address the question of evidence directly, but they are still centralized. They assume that accurate ratings can be assumed and effectively aggregated. Such ratings are typically just scalars (possibly augmented with free text comments). However, trust is often multidimensional. Moreover, the best ratings are those that are obtained privately and from parties that you know personally. We take this idea and formalize it into the notion of referrals, where a referral can be viewed as an endorsement.

* This research was supported by the National Science Foundation under grant ITR-0081742. We thank the anonymous reviewers for helpful comments.

© Springer-Verlag Berlin Heidelberg 2004

On the theoretical side, studies of Web structure have also assumed the links from one to page to another indicate some sort of an endorsement. This assumption leads to the heuristic that a metric such as the PageRank of a page measures its authoritativeness [3, 4]. Informally, a Web page has a high PageRank only if it is pointed to by Web pages with high PageRanks, i.e., if other authoritative pages view this page as authoritative. Current work on Web structure generally mines the Web and then studies graphs induced by links among Web pages. It offers statistical models of the Web and its evolution, e.g., [2, 13], but doesn't capture the reasoning by each creator of a link, through which a link can be taken to be an endorsement.

Fortuitously, our referral-based approach for finding trustworthy service providers gives us a direct means to model the evolution of a "social" network of agents. We can consider the various representations and strategies or policies used by the agents along with distributions of true authoritativeness and relate these to each other. Our study of the processes of linkage and referrals offers some key benefits:

- It applies in helping determine trust in dynamic settings.
- It gives us a basis for understanding how trust and authoritativeness emerge, potentially leading to mechanisms that cannot be easily violated.
- The above mechanisms can be used to develop measures of authoritativeness and trust that apply to settings such as the Deep Web, where Web sites respond to queries, but do not carry statically linked pages.

Our study is conducted as a series of simulation experiments where we can control the policies of the agents. These experiments consider up to several hundred agents. Although this is far from the size of the Web, the simulations can still offer some understanding of the concepts and processes involved. The immediate practical applications will relate to enterprise knowledge management and virtual communities, where sizes of a few hundred to a few thousand are common.

Organization. Section 2 is the technical background and is largely based on our previous papers [17]. It provide details on our model of referrals among autonomous agents, the application domain, and our experimental setup. Section 3 studies factors that affect the distribution of PageRanks. Section 4 studies the ways PageRank distributions can evolve based on neighbor selection policies. Section 5 discusses the relevant literature and motivates directions for further work.

2 Technical Framework

We consider multiagent systems consisting of autonomous agents. Agents represent *principals* who could be people or businesses providing and consuming *services*. The services are construed abstractly, i.e., not limited to current Web services standards. Specifically, the services could involve serving static pages, processing queries, or carrying out e-commerce transactions, but their details are not represented in this paper. Our study will concentrate on agents and services.

The agents offer varying levels of trustworthiness and are interested in finding other trustworthy agents. An agent begins to look for a trustworthy provider for a desired service by querying some other agents from among its *neighbors*. The neighbors of an agent are a small subset of the agent's acquaintances that have been useful before.

The agents are autonomous. That is, a queried agent may or may not respond to another agent by providing a service or a referral. The querying agent may accept a service offer, if any, and may pursue referrals, if any. When an agent accepts a service or follows a referral, there are no guarantees about the quality of the service or the suitability of a referral. We do not expect that any agent should necessarily be trusted by others: an agent decides how to rate another principal based on its own means. Notice that trust applies both to the ultimate service provider and to the agents who contribute to referrals to that provider.

Each agent maintains models of its acquaintances, which describe their *expertise* (i.e., quality of the services they provide) and *sociability* (i.e., quality of the referrals they provide). Both of these elements are learned based on service ratings from its principal. Using these models, an agent decides on which of its acquaintances to keep as neighbors. Key factors include the quality of the service received from a given provider, and the resulting value that can be placed on a series of referrals that led to that provider. In other words, the referring agents are rated as well.

The above model addresses the important challenge of finding trustworthy agents, which is nontrivial in open systems. One, referrals can apply even in the absence of centralized authorities and even when regulations may not ensure that services are of a suitable quality. Two, because service needs are often context-sensitive, a response from an agent can potentially benefit from the knowledge that the agent has of the other's needs.

2.1 Application Domain

We apply the above framework in a commerce setting, where the service providers are distinct from the service consumers. The service consumers lack the expertise in the services that they consume and their expertise doesn't get any better over time. However, the consumers are able to judge the quality of the services provided by others. For example, you might be a consumer for auto-repair services and never learn enough to provide such a service yourself, yet you would be competent to judge if an auto mechanic did his job well. Similarly, the consumers can generate difficult queries without having high expertise. For example, a consumer can request a complicated auto-repair service without having knowledge of the domain.

Figure 1 is an example configuration of service consumers and providers that corresponds to a commerce setting. The nodes labeled C denote consumers and the nodes labeled S denote service providers. Consumers are connected to each other as well as to the service providers. These links are essentially paths

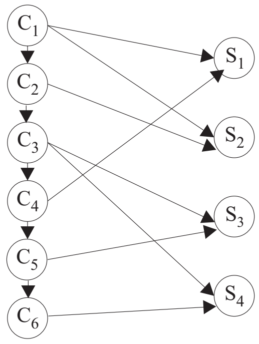


Fig. 1. A schematic configuration for e-commerce

that lead to service providers with different expertise. In this model, the service providers are dead ends: they don't have outgoing edges, because they don't initiate queries or give referrals. Thus, their sociability stays low. Their true and modeled expertise may of course be high.

2.2 Agent Algorithms

We have implemented a distributed platform using which adaptive referral systems for different applications can be built. However, we investigate the properties of interest over a simulation, which gives us the necessary controls to adjust various policies and parameters.

Consumers have high *interest* in getting different types of services, but they have low expertise, since they don't offer services themselves. Providers have high expertise but low sociability. The interests and expertise of the agents are represented as term vectors from the vector space model (VSM) [15], each term corresponding to a different domain. The simulation uses these term vectors to generate queries and answers for the various agents.

Each agent is initialized with the same model for each neighbor; this initial model encourages the agents to both query and generate referrals to their neighbors. An agent that is looking for an answer to a query follows Algorithm 1. An agent generates a query by slightly perturbing its interest vector, which denotes that the agent asks a question similar to its interests (line 1). Next, the agent sends the query to a subset of its neighbors (line 2). The main idea here is to determine which of its neighbors would be likely to answer the query. We determine this through the *capability* metric.

The capability of an agent for a query measures how similar and how strong the expertise of the agent is for the query [16]. Capability resembles cosine similarity but also takes into account the magnitude of the expertise vector. What this means is that expertise vectors with greater magnitude turn out to be more capable for the query vector. In Equation 1, Q ($\langle q_1 \ldots q_n \rangle$) is a query vector, E ($\langle e_1 \ldots e_n \rangle$) is an expertise vector and n is the number of dimensions these vectors have.

Algorithm 1 Ask-Query()

```
1: Generate query
2: Send query to matching neighbors
3: while (!timeout) do
4:    Receive message
5:    if (message.type == referral) then
6:       Send query to referred agent
7:    else
8:       Add answer to answerset
9:    end if
10: end while
11: for $i = 1$ to $|answerset|$ do
12:    Evaluate answer($i$)
13:    Update agent models
14: end for
```

$$Q \otimes E = \frac{\sum_{t=1}^{n} (q_t e_t)}{\sqrt{n \sum_{t=1}^{n} {q_t}^2}} \tag{1}$$

An agent that receives a query provides an answer if its expertise matches the query. If it does, then the answer is the perturbed expertise vector of the agent. When an agent does not answer a question, it uses its *referral policy* to choose some of its neighbors to refer.

Back in Algorithm 1, if an agent receives a referral to another agent, it sends its query to the referred agent (line 6). After an agent receives an answer, it evaluates the answer by computing how much the answer matches the query (line 12). Thus, implicitly, the agents with high expertise end up giving the correct answers. After the answers are evaluated, the agent updates the models of its neighbors (line 13). When a good answer comes in, the modeled expertise of the answering agent and the sociability of the agents that helped locate the answerer (through referrals) are increased. Similarly, when a bad answer comes in, these values are decreased. At certain intervals during the simulation, each agent has a chance to choose new neighbors from among its acquaintances based on its *neighbor selection policy*. The number of neighbors is limited, so if an agent adds some neighbors it drops some neighbors as well.

Together, the neighborhood relations among the agents induce the structure of the given society. In general, as described above, the structure is adapted through the decisions of the different agents. Although the decisions are autonomous, they are influenced by various policies.

3 Distribution of PageRanks

It has been widely accepted that the in-degree and out-degree distributions on the Web follow a power-law distribution [2]. That is, the number of pages with k

incoming links is inversely proportional to k^m; Barabási *et al.* estimate that $m = 2.1$ [2]. Recently, Pandurangan *et al.* [12] showed that the distribution of the PageRanks on the Web follows a power-law as well. Here, we study four factors that influence the PageRank distributions.

The experiments we report below contain 400 agents, where each agent is neighbors with four other agents. The initial neighbors are picked at random. All the populations contain 400 agents. Each service provider has a high expertise in one domain. The remaining agents are service consumers. The interests of the service consumers can span multiple domains. We tune the simulation so that an agent answers a query only when it is sure of the answer. This ensures that only the providers answer any questions, and the consumers generate referrals to find the providers.

The PageRank of a Web page measures its authoritativeness. Informally, a Web page has a high PageRank only if it is pointed to by Web pages with high PageRanks, i.e., if other authoritative pages view this page as authoritative. Intuitively, the same metric can be applied to referral networks to measure the authoritativeness of agents. In the case of referral networks, an agent would be considered authoritative if it has been pointed to by other authoritative agents. Recall that an agent is pointed to by other agents if it is providing useful answers or referrals. Hence, if an authority finds another agent useful and points at it, then this agent is considered a good authority as well. This way, agents in the decide on who is authoritative in the referral network.

The PageRank of an agent is calculated using Equation 2, where $P(i)$ denotes the PageRank of agent i, I_i denotes agents that have i as a neighbor, and N_j denotes the agents that are neighbors of j. In addition to accumulating PageRanks from incoming edges, each node is assumed to get a minimum PageRank of $(1 - d)$.

$$P(i) = d \sum_{j \in I_i} \frac{P(j)}{|N_j|} + (1 - d) \tag{2}$$

For our calculations, we pick d to be 0.85. There are two reasons for this. One, 0.85 is the value mostly used and suggested for PageRank calculations on the Web [3]. Two, a high value denotes that an agent receives most of its PageRank from the agents that point at it, rather than a given default value. For example, if d was picked to be 0, each page would receive an equal PageRank and thus the link structure would not have affected the PageRank of the agents. The calculated PageRanks are not normalized to demonstrate the variance in maximum PageRanks in different setups.

The PageRanks of the agents are calculated centrally by building a graph from the neighborhood relations after the simulations. We study how the percentage of actual experts in the network, the referral policies that the agents follow, adaptability of the agents, and the neighbor selection policies they follow affect the PageRank distributions.

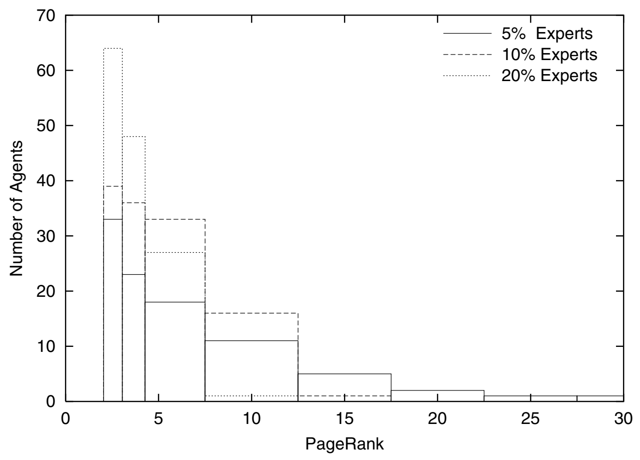


Fig. 2. PageRank distributions for percentage of experts

3.1 Percentage of Experts

Intuitively, the percentage of agents with high expertise plays a crucial role in the distribution of PageRanks. For example, when there are too many experts in the system, we expect that the PageRanks will tend to be shared among them. Having a small number of experts may ensure that experts with high authoritativeness will emerge. To study this point, we vary the percentage of the experts in the system. We study three populations with 5%, 10%, and 20% experts in them.

Figure 2 shows a histogram of PageRank distribution for three populations for PageRank values 2.5 and higher. The solid lines denote the population with 5% experts, the dashed lines denote the population with 10% percent experts, and the dotted lines denote the population with 20% experts. When the percentage of experts is high, the PageRanks are clustered for small PageRank values. For example, when the population has 20% experts, the number of agents having PageRank higher than 2.5 is more than the cases for the other two populations. For the higher values of the PageRank, the converse holds. For example, the only population that allows PageRanks higher than 25 is the 5% expert population.

An intuitive explanation for this phenomenon is the implicit competition among the experts. When there are too many of them, they end up sharing the incoming edges thereby only a few reach relatively high PageRanks. When there are a few experts, those experts tend to dominate more clearly. Since the population with 5% percent experts provide a broader distribution of PageRanks, we use this population for the following experiments.

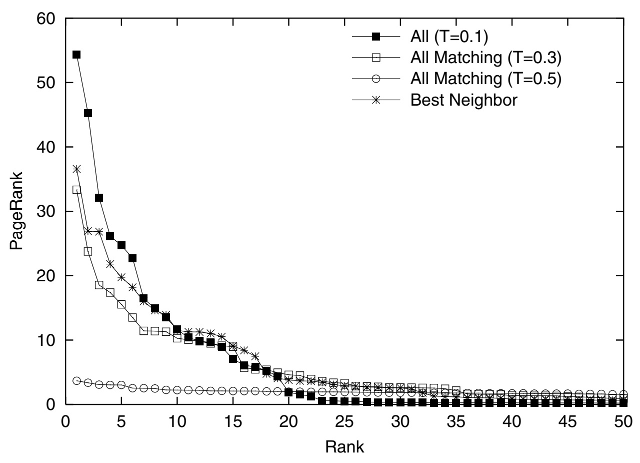


Fig. 3. PageRank distributions for referral policies

3.2 Referral Policies

A referral policy specifies to whom to refer. We consider some important referral policies. Based on the referral policy, each query results in a different number of agents being contacted. We limit the length of the referral chains to three—similar to Gnutella's time-to-live value.

1. *Refer all neighbors*. Agents refer all of their neighbors. This policy resembles Gnutella's search process where each node forwards an incoming query to all of its neighbors if it doesn't already have the requested information [6].
2. *Refer all matching neighbors*. The referring agent calculates how capable each neighbor will be in answering the given query (based on the neighbor's modeled expertise). Only neighbors scoring above a given capability threshold are referred.
3. *Refer the best neighbor*. Agent refer the best matching neighbor. This is similar to Freenet's routing of request messages, where each Freenet client forwards the request to an agent that is the likeliest to have the requested information [9].

Some agents are identified as authoritative as a result of their being chosen as neighbors by other authoritative agents. Presumably, authoritative agents are the most desirable to interact with. In general, agents with high expertise or high sociability would be candidates for becoming authorities. We measure the authoritativeness of each agent using the PageRank metric (Equation 2) and study the effect of referral policies in the emergence of authorities.

After each simulation run, the agents are ranked based on their final PageRank. Figure 3 shows the PageRank distribution of the top 50 agents (out of a total of 400). If the agents use the *Refer all* policy, few authorities with high PageRanks emerge. For example, the 10th agent in the *Refer all* policy gets a

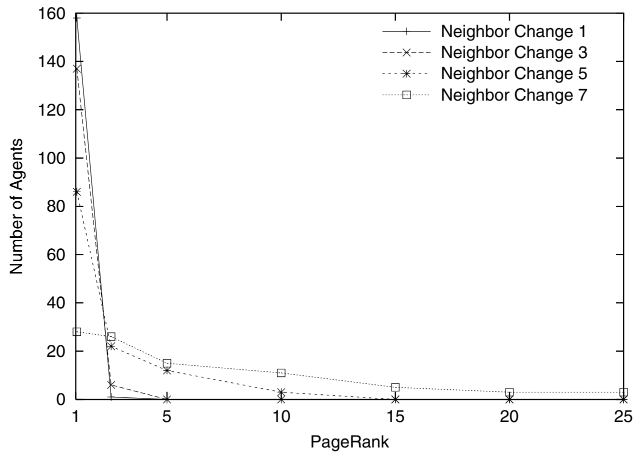


Fig. 4. Change in PageRank distribution as agents adapt

PageRank greater than the first agent in two instances of the *Refer all matching* policy (with thresholds 0.4 and 0.5). Further, the *Refer all* policy creates a large variance among the PageRank distributions. For example, while the first agent gets a PageRank of 54.00, the 50th agent gets a PageRank of only 0.23.

Contrast this with *Refer all matching* policy with a threshold of 0.5, where the first agent gets a PageRank of 3.68 whereas the 50th agent gets a PageRank of 1.58. The distribution of PageRanks using the *Best neighbor* policy falls between the distributions for *Refer all* and *Refer all matching* with high thresholds. In other words, when agents use the *Best neighbor* policy, the highest PageRank is not as high as the *Refer all* policy (36.00) but the PageRank variance between the first and the 50th agent is still quite large.

Intuitively, the explanation for the above is that the *Refer all* policy is highly effective in disseminating information about the experts. With *Refer all* policy, the consumers locate the providers easily. When this policy is used, most consumers have in-degree zero, and the possible in-degrees are shared among the providers almost equally.

3.3 Adaptability

At certain intervals during the simulation, each agent has a chance to choose new neighbors from among its acquaintances. Since agents learn about other agents through the answers they receive, changing neighbors allow them to point at agents that are expected to be more useful to them. This allows us to study the evolution of PageRanks, since PageRanks of the agents change as a result of the neighbor changes.

Figure 4 plots the distribution of PageRanks after several neighbor changes. The plots correspond to the number of agents that get a PageRank higher than the value given on the X axis. To reduce clutter, we have omitted the PageRanks smaller than 1. After the first neighbor change, most agents have a PageRank between 1 and 3. As the agents change their neighbors and point to the ones that are more useful to them, some authorities emerge. This is reflected in the graph by the increased number of agents that get a higher PageRank. After the final neighbor change, there are agents with PageRanks greater than 25. In other words, the agents do learn about the authorities and show a preference for linking to them.

3.4 Neighbor Selection Policies

Recall that each agent chooses its neighbors based on local information only, without knowing which neighbors other agents are choosing. A neighbor selection policy governs how neighbors are added and dropped. Such policies can strongly influence the formation of authorities.

To evaluate how the neighbor selection policies affect the PageRank distribution, we compare three policies using which an agent selects the best m of its acquaintances to become its neighbors.

– *Providers*. Sort acquaintances by how their expertise matches the agent's interests.
– *Sociables*. Sort acquaintances in terms of sociability.
– *Weighted average*. Sort acquaintances in terms of a weighted average of sociability and how their expertise matches the agent's interests.

Figure 5 plots the distribution of PageRanks with respect to some neighbor selection policies. Again, the X axis shows PageRanks and the Y axis denotes the number of agents that get a PageRank greater than the PageRank shown on the X axis. The five plots correspond to *Providers*, *Sociables*, and three *Weighted average* neighbor selection policies with different weights. W denotes the weight of the sociability in choosing a neighbor. When W is set to 0, the Providers policy, and when W is set to 1, the Sociables policy is in effect. Other values of W measure weighted averages of the sociability and expertise.

All curves, except the one for *Sociables* policy, are similar to each other. In all four cases, only a few number of authorities emerge. But, the level of their authoritativeness is high. For example, for the *Providers* policy, while only 26 agents get a PageRank above 1, five of them get a PageRank above 20. Increasing the effect of the sociability slightly increases the number of agents with medium authority while slightly decreasing the number of agents with high authority. For example, with *Weighted Average* policy, when the sociability and the expertise are weighted equally, the number of agents that get a PageRank above 1 is 44, while four of them get a PageRank above 20.

The *Sociables* policy does not follow this distribution. Initially, when not too many experts have been discovered, choosing neighbors only based on sociability does not help agents find service providers. Hence, when agents follow the *Sociables* policy in the beginning, most agents get average PageRanks (e.g., 158

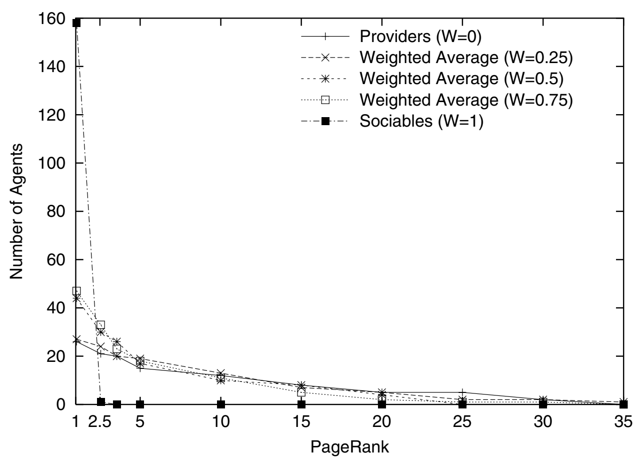


Fig. 5. PageRank distributions for neighbor selection policies

agents get a PageRank around 1). However, once the PageRanks distribution settles, following *Sociables* policy instead of *Providers* policy determines how the PageRank distribution evolves.

In other words, for strong authorities to emerge, it is important that the agents put a high value on the ability to produce high quality of service. If at the outset, the agents prefer sociables, there is little grounding in quality, it is difficult to find good providers and strong authorities do not emerge. However, once the network has stabilized, sociability helps as there is a basis for good referrals to be given and there is value in those who can give good referrals.

4 Evolution of PageRank Distributions under Perturbation

The factors mentioned in the previous section influence the PageRank distributions. As expected, the agents that get the most PageRanks are agents with high expertise or high sociability. Now, we manually modify the models of a few of the agents in the system to measure the consequences of the perturbation (vis à vis the existing PageRanks, which characterize the entrenched players) on the resulting population.

For example, what happens when an agent with a high expertise start providing poor quality of service? Conversely, if an agent gains expertise over time, can it acquire high PageRanks and get a high ranking?

First, we run the simulation until the neighborhood graph stabilizes, i.e., there are no more neighbor changes. Then, we pick four agents that have ranked in the top 25 and switch their expertise values with four agents with low expertise. This will ensure that the four previously expert agents will start giving low quality service. Rather than choosing the top four agents, we choose the four

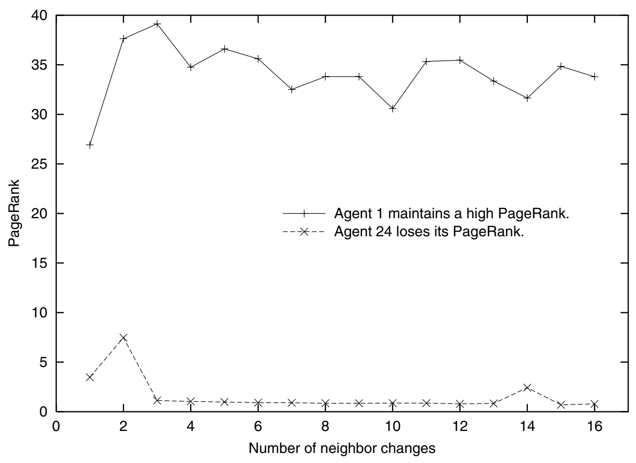


Fig. 6. Evolution of PageRanks for two previous experts

agents uniformly from the top 25 (1st, 4th, 16th, and 24th). This will allow us to see the impact of the initial position in the rankings.

We are interested in how the PageRank distribution and the ranking of these eight agents will change. Intuitively, we would expect the PageRanks to drop for the previously expert agents and increase for the agents who have now become experts. To give a more complete analysis, below we look at two orthogonal setups. In the first setup, agents use a neighbor selection policy that favors sociability, whereas in the second setup the agents use a neighbor selection policy that favors expertise.

4.1 Agents Prefer Sociables

We first study the four agents that have started giving low quality of service. Figure 6 plots the evolution of PageRanks for two previous experts, Agent 1 and Agent 24. Interestingly, when agents prefer to be neighbors with sociable agents, the PageRanks of the two agents (4th and 24th) get a lot more affected than the other two agents. More precisely, the PageRanks as well as the rankings of these two agents drop.

The agent that has ranked 4th is now ranked at 18 and the agent that was ranked 24th is now ranked at 48. However, the other two agents (1 and 16) do not get affected by the fact that they have started offering lower quality of service. Both their PageRank and ranking stay almost the same with minor shifts between neighbor changes. This is unexpected. Looking closely at the agents that model these, we see that even though agents 1 and 16 have low expertise, they are still being kept as neighbors for their sociability. In other words, now that agents 1 and 16 cannot provide adequate answers, they start giving good referrals. This is enough for other agents pointing at these to keep

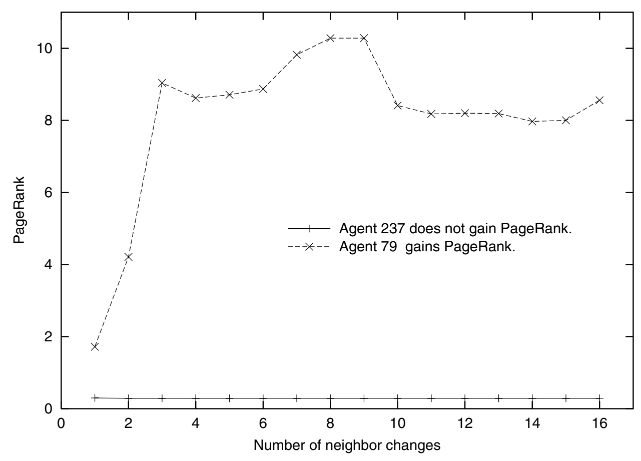


Fig. 7. Evolution of PageRanks for two new experts

them as neighbors. In other words, agent 1 and 16 keep their PageRanks because of their neighbors. On the other hand, since agents 4 and 24 do not have as useful neighbors, there is no use for other agents to point at them. What we see here is that when agents gain enough PageRank, they can maintain their ranking even after they lose their expertise as long as they maintain useful neighbors.

Meanwhile, the four agents who have now started offering good services can increase their PageRanks only slightly; the greatest increase was of PageRank 1. So, even though these agents have high expertise now, since the other agents care for sociability more, these four agents are not being picked up as neighbors as much as expected.

4.2 Agents Prefer Experts

The previous section explained how the PageRank distribution can evolve when the agents prefer sociable neighbors. In this section, we look at the converse case: how does the distribution evolve if the agents choose experts over sociables?

Again, we look at the four agents who have stopped providing high quality of service. This time, all of their PageRanks, and hence their ranking, drop slowly. The interesting cases occur for the other four agents who have now started providing high quality of service. Figure 7 plots the evolution of PageRanks for two of the agents.

Agent 237 cannot improve its PageRank at all. The reason for this is that only a few other agents point at it. Therefore, it is not referred to other agents. Over all, even though this agent has higher expertise than most of the agents that rank above it, it cannot get enough PageRank to improve its ranking.

This is not the case for agent 79. It gets discovered by other agents and hence can substantially increase its PageRank. However, there is a limit to this increase. That is, even though it accumulates a high PageRank, it cannot make it to the top ten. This is caused by the stability of the other agents' neighbors. Most other agents have already found good service providers for their needs, so they rarely change neighbors.

4.3 Does the Winner Take All?

The power-law distributions on the Web suggest some sort of *preferential attachment* wherein Web links are likelier to be made to those pages that already have a high in-degree or PageRank. Hence, the mottos "the rich get richer" and the "winner takes all" are used to describe this phenomenon. However, Pennock *et al.* recently discovered that the reality is closer to a combination of preferential and uniform attachment, meaning that the winner doesn't always take all [13].

Note that the traditional Web approaches seek to model the Web without regard to an understanding of why links emerge. These models are limited to static pages where the links are generated by hand and any semantics is hidden in the surrounding text. By contrast, we are interested in and able to model the *process* through which the links among agents emerge. When the agents contact others or generate referrals, they do so with respect to specific service requests. Further, we can explicitly model the various policies through which the referrals are generated and through which the agents revise their choice of neighbors. This enables us to capture various shades of trust formally.

From the above experiments, we conclude that authorities with high PageRanks emerge when (1) there are few experts in the system, (2) agents exchange more referrals, and (3) agents change neighbors based on other agents' expertise and sociability. The PageRank distribution then evolves based on how these authoritative agents keep up with their quality of service. This evolution is highly influenced by whether agents prefer sociables or experts. When agents prefer sociables, agents who have gained high PageRanks can sometimes keep their PageRank even without providing high quality of service. On the other hand, when experts are chosen over sociables, the agents with high PageRanks have to continue to offer high quality of service.

Similarly, when sociables are preferred, there is little chance for newcomer experts to get high PageRanks since they are not pointed to by any sociable agents. Preference for experts relaxes this case but still does not guarantee high PageRanks for newcomers. These findings lead us to conjecture more broadly that in environments where sociability dominates (with respect to expertise), the winner will take all, whereas in settings where expertise dominates, the winner may not take all. If new services emerge and draw some of the demand, then there will be greater churn in the rankings of the top most authorities.

5 Discussion

Our approach enables us to study the emergence of link structure of the Web as agents give one another referrals. The link structure evolves based on agents' preferences in neighbor choices (i.e., expertise or sociability). The dynamic nature of the link structure helps us study the emergence of authorities, rather than merely identify the authorities on a static graph. Below, we discuss some related approaches.

Referrals have been used in multiagent systems before, but the emergent nature of referral network has not received enough attention. For example, MINDS, which helped users find documents, was an early agent-based referral system [5]. MINDS system allows adaptivity but the self-organization of the nodes have not been studied. Kautz *et al.* study some properties of static graphs that result from the referrals exchanged among agents, but do not have adaptivity and hence have no self-organization [7].

Kumar *et al.* develop an approach to infer web communities from the link structure of the Web [8]. Kumar *et al.* propose that any community structure should contain a bipartite core where the *fans* and *centers* make up the independent sets. Fans and centers are defined recursively, such that fans are pages that point at good centers and centers are pages that are pointed to by good fans. Kumar *et al.*'s approach assumes that if many fans point to the same set of centers, then they are likely to be on the same topic, and hence form a community.

Pujol *et al.* calculate the reputation of an agent based on its position in its social network [14]. The social networks are built based on the link structure induced by the web pages of the users. An agent gets a high reputation only if the agents that point to it also have high reputation, similar to the notion of authority exploited in search engines such as Google. Pujol *et al.* test their approach to find the reputations of authors where the reputation of an author is defined as the number of citations received. Even though each agent can calculate its own reputation based only on local information (i.e., the agents that point at it), a central server is needed to access others' reputations.

Adamic *et al.* study different local search heuristics that exploit high outdegrees in power-law networks [1]. For example, one of their heuristics is based on sending the message to the neighbor with the most out-degree, assuming each node is aware of the number of outgoing edges of their neighbors. This is similar to our concept of sociability. In their approach, a peer with high out-degree is chosen because it will allow the message to get to more peers. In our case, rather than maximizing the nodes that will receive the query, we try to send the query only to those who are likely to answer. For this reason, finding a sociable agent is important since the referrals it gives will help locate an expert.

Ng *et al.* study the stability of PageRank and HITS algorithms and propose variants of these algorithms that offer more stability with respect to the addition or deletion of vertices [10] . They take a graph-theoretic stance and simply recalculate the given metrics. Their results are of course valuable in judging the stability of the different algorithms. By contrast, however, we consider the processes explicitly. Thus we can address the question of how the links will evolve.

Overall, we believe that a process-centric view doesn't detract from the previous studies, but adds additional depth and understanding to how networks evolve in cooperative settings, but under local control. It also suggests how to apply various policies to maximize the quality of a given network of agents. There is an immediate value for the smaller networks, for instance, for knowledge management by locating experts, e.g., [18], and a longer-term value in understanding the evolving Web at large.

References

[1] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Physics Review E*, 64(46135), 2001. 209

[2] Albert-László Barabási, Réka Albert, and Hawoong Jeong. Scale-free characteristics of random networks: The topology of the World Wide Web. *Physica A*, 281:69–77, 15 June 2000. 196, 199, 200

[3] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998. 196, 200

[4] Monika R. Henzinger. Hyperlink analysis for the Web. *IEEE Internet Computing*, 5(1):45–50, January 2001. 196

[5] Michael N. Huhns, Uttam Mukhopadhyay, Larry M. Stephens, and Ronald D. Bonnell. DAI for document retrieval: The MINDS project. In Michael N. Huhns, editor, *Distributed Artificial Intelligence*, pages 249–283. Pitman/Morgan Kaufmann, London, 1987. 209

[6] Gene Kan. Gnutella. In *[11]*, chapter 8, pages 94–122. 2001. 202

[7] Henry Kautz, Bart Selman, and Mehul Shah. ReferralWeb: Combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65, March 1997. 209

[8] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Extracting large-scale knowledge bases from the Web. In *Proceedings of the 25th Very Large Databases Conference*, pages 639–650. Morgan Kaufmann, 1999. 209

[9] Adam Langley. Freenet. In *[11]*, chapter 9, pages 123–132. 2001. 202

[10] Andrew Y. Ng, Alice X. Zheng, and Michael I. Jordan. Link analysis, eigenvectors and stability. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 903–910, 2001. 209

[11] Andy Oram, editor. *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*. O'Reilly & Associates, Sebastopol, CA, 2001. 210

[12] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Using PageRank to characterize Web structure. In *8th Annual International Computing and Combinatorics Conference (COCOON)*, pages 330–339, 2002. 200

[13] David Pennock, Gary Flake, Steve Lawrence, Eric Glover, and C. Lee Giles. Winners don't take all: Characterizing the competition for links on the Web. *Proceedings of the National Academy of Sciences*, 99(8):5207–5211, April 2002. 196, 208

[14] Josep M. Pujol, Ramon Sangüesa, and Jordi Delgado. Extracting reputation in multi agent systems by means of social network topology. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 467–474. ACM Press, July 2002. 209

[15] Gerard Salton and Michael J. McGill. *An Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983. 198

[16] Munindar P. Singh, Bin Yu, and Mahadevan Venkatraman. Community-based service location. *Communications of the ACM*, 44(4):49–54, April 2001. 198

[17] Pınar Yolum and Munindar P. Singh. Emergent properties of referral systems. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 592–599. ACM Press, July 2003. 196

[18] Bin Yu and Munindar P. Singh. Searching social networks. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 65–72. ACM Press, July 2003. 210

Adaptiveness in Linda-Based Coordination Models

Ronaldo Menezes[1] and Robert Tolksdorf[2]

[1] Florida Institute of Technology, Department of Computer Sciences
150 W. University Blvd., Melbourne, FL 32901, USA
rmenezes@cs.fit.edu
http://cs.fit.edu/~rmenezes

[2] Freie Universität Berlin, Institut für Informatik
AG Netzbasierte Informationssysteme
Takustr. 9, D-14195 Berlin, Germany
research@robert-tolksdorf.de
http://www.robert-tolksdorf.de

Abstract. In today's software engineering, the ability to adapt to requirement changes on-the-fly has become a major asset to computer systems. Traditionally, systems were developed following a process consisting of: requirements study, design and implementation phases. Nowadays, requirements are very dynamic – changing constantly. For the engineering of distributed complex software systems, the frameworks used in the implementation phase need to be able to cope with environment changes – bandwidth fluctuations, network topology, host down time, etc. This paper looks at the ability of Linda-based systems to adapt to the predictable and unpredictable changes in distributed environments. Several adaptiveness aspects are discussed and a few of Linda-based systems analyzed.

1 Introduction

Software engineers are constantly faced with the hard task of developing systems that can adapt to changes in computational environments. With concepts such as mobility becoming commonplace, the job of designing even a "simple" client-server system is not as trivial as it used to be. In mobile environments, the clients can dynamically change their location and so can the servers. Therefore standard techniques such as socket communication may not be applicable.

Coordination models such as Linda [11] advocate that distributed systems are better designed if one can think about the control of dependencies between computational components in isolation. That is, coordination between distributed components can (and should) be dealt separately from the computational task(s) the ensemble is trying to perform. Unfortunately, the need for ever-more-complex applications makes the current *status quo* in coordination systems inappropriate to deal with the intricacies of these systems as well as with the dynamism of the environments they are implemented on.

© Springer-Verlag Berlin Heidelberg 2004

The word here is *adaptiveness*. Traditionally, we developed systems following the well define phases of requirements study, design and implementation. More often than not, the requirements study for today's systems is incomplete or is not able to predict all the possible environment changes. In the last few decades we have seen the advent of several specialized areas trying to deal with changes in distributed systems environments. We have seen research on fault tolerance attempt to deal with unpredictable and predictable failures in parts of the environment (eg. nodes being down, network links being lost). We have also witnessed the advent of mobile computation as a mechanism to deal with the movement of data and processes as an attempt to improve communication latency – Cardelli [5] has argued that mobile computation is a way to "cheat" communication latency.

One cannot deny that the areas above have had relative success. Fault-tolerant systems are a reality and so is mobile computation. However, it is not uncommon to see the research in these areas trying to isolate their problems. That is, research on fault-tolerance is likely to disregard the fact that entities may be mobile and research on mobile computation may disregard the fact that failures are frequent.

The scenario above can also be observed in coordination systems such as the ones based on the Linda model. Early Linda models of coordination were neither very expressive nor able to deal with the dynamism of today's applications [16]. Throughout the years several extensions and new models have been proposed to deal with some of the problems mentioned above but as expected they normally focus on one issue. A few important extensions to the original Linda model are: MTS-Linda [12], Laura [26], KLAIM [9], PLinda [1], TuCSoN [20], Lime [22], LogOp [24], PeerSpaces [4] and TOTA [15]. These models tackle from simple things such as separations of concerns to more interesting problems such as mobility and fault-tolerance.

This paper discusses several problems such as fault-tolerance and mobility awareness as a specialization of a more general problem that we call *adaptiveness-awareness*. We argue that an adaptive model should be able to cope with these and other problems in a clean and simple way without having to burden the users and developers with details that, truly, don't concern them. Next, we discuss in Section 2 adaptiveness and some of its aspects. Section 3 discusses how some of the models above cope with the issues raised in Section 2. Then, Section 4 describes the concept of *swarming* as an adaptive mechanism and demonstrates how its concepts can be thought in terms of Linda, given rise to a model we call SwarmLinda [27]. Last, some discussion on issues related to SwarmLinda is presented in Section 5.

2 Aspects of Adaptiveness

The ability to be adaptive is becoming a major benefit to any open, distributed system. Adaptiveness is important for several reasons: our inability to predict future problem-solving workloads, future changes in existing information requests, future failure and additions of resources, and future environment characteristics that require system organization and future migration of data and agents [10].

In the context of distributed systems, the inability to predict future workloads relate to network problems such as unpredictable bandwidth fluctuations and network latencies as well as bottlenecks created due to overload of particular network nodes.

Changes in patterns of information request reflect directly in distributed systems. Agents need to be able to store and search for information efficiently. Historically, systems dealt with this issue at design time (eg. by choosing a static distribution mechanism) but the dynamism of today's applications causes these patterns to be unpredictable.

Openness is not a desirable feature but a requirement to most large scale distributed systems. The ubiquity of the Internet and its availability as a framework for distributed applications makes open system issues the rule instead of being the exception. Software engineers need to consider the fact that resources may be added or removed at runtime. Removal may also occur due to resource (eg. network node) failures.

If the above was not sufficient to the life of a software engineer, mobility is very common as a mechanism to improve the system is efficiency. If resources can move "closer" to their requestors, the system as a whole can only gain. However, not only the patterns of requests are unpredictable, but the mobility itself make the design harder.

We now discuss these issues as facets in adaptiveness of distributed systems.

2.1 Network Unpredictability

The first aspect of adaptiveness that needs to be considered is the ability of a system to handle unpredictable network behavior. Ideally, we would like to engineer systems that can make efficient use of the Internet but its unpredictable behavior and lack of quality of service (QoS) makes the engineering of any practical application very hard.

In the engineering of distributed systems, we are constantly faced with the issue of how to distribute the information required by processes in such a way that a fair load-balancing between all the nodes is achieved. The problem here is that one cannot base an information distribution scheme on static information about how "close" the nodes are from each other. Closeness in networks is measured in delays and not in units of geometric distance, thus what can be seen as "close" at design time may become very "far" as the application is deployed.

Systems dealing with complex systems must attempt to maintain the "closeness" characteristics in spite of any network unpredictability. One way to achieve this would be by finding different routes between source and destination. Alternatively information can be made mobile and be kept "close" to where it is most required (as described in the Mobility Section below).

All in all, engineers need to be freed from the pointless task of identifying the best distribution of information. Even the best engineers cannot accurately predict the status of a wide networks such as the Internet.

2.2 Failures

Disturbingly, failures are more common than one might think – they are not exceptional (special) cases. Most models of coordination opt to unrealistically assume that the environment used by the applications is failure-free. Of course there are fault-tolerant coordination models such as PLinda [1] but even these have a very simplistic view of what failures are and, worse, they use solutions that are costly. For instance, PLinda uses a transaction mechanism to guarantee that a set of operations either take place or not (transactional semantics). The implementation of transactions across a fully distributed environment may be very expensive as it involves distributed locking mechanisms.

In [25] we studied mechanisms to provide fault-tolerance in Linda systems. The approaches taken fall mostly into the following categories: transactions, mobile coordination, replication and checkpointing. All these mechanisms are rather static in that they assume a fixed location of faults and fixed locations of where faults are managed. Also, they take a strict approach on failures: either the system is in a state of failure – in which case it is managed to perfection – or it is without failure. Any intermediate state – like partially inconsistent replicas – is not considered a valid state of the system.

The approach proposed in [23] adds slightly more flexibility by using mobile code. Computations are transfered to a server where it accesses a tuple-space. If the computation fails there, a "will" transmitted together with the code is executed after the death of the computation. Thereby at least some application-level flexibility is added to static reactions of failures.

Here we want to see failures as a natural phenomenon in large distributed systems running on unreliable networks such as the Internet. A node failure should have minimum effect to the application behavior. A severe effect is, however, entering a state of exclusive managing the failure situation. Therefore, allowing partial failures and incremental working towards a fault-free state seems a more suitable approach. Self-organization is an incremental mechanism which does not establish an optimal state in one step but rather works towards it.

2.3 Mobility

Another feature that is becoming more common is the ability of agents to be mobile within the environment they work. This is in addition to an older form of mobility that is data (passive information) mobility. The motivation for mobility is primarily performance. By allowing passive and active entities to roam around the network, one can attempt to make requestor and requested closer to each other thus minimizing communication overhead.

Let's first look into passive data mobility. The result of allowing passive data to hop from place to place in the network is that applications need to be able to find this information. Any piece of information closer to an agent is useless if the agent does not know where it is! What we're describing here is that agents need to be able to adapt to changes in the location of the data they need – another aspect of adaptiveness.

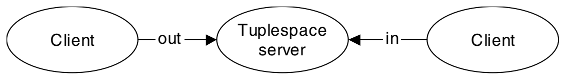


Fig. 1. Centralized tuple spaces

Conversely, the environment needs to know where the requests are coming from. If agents are mobile they may request some information while at location X and then migrate to location Y before the information is received. How can the environment know where the agent is? How can the information requested find the agent?

The simple scenarios above show that modern distributed applications need to exhibit some level of adaptiveness to be able to cope with on-the-fly changes in the environment.

A different approach to mobility is physical mobility as in mobile computing (ie. PDAs, notebooks, smart phones, etc.) as opposed to mobile computation (described above). This approach adds a different mobility level to any system. One can have data mobility (as above) added to the physical mobility of nodes, or have them independently. The movement of nodes causes the topology of the network to change. If physical mobility is a common characteristic then systems need to be able to adapt to the changes in the network topology without which they may be incapable of finding the information they require.

2.4 Openness

How big is enough? How small is just right? This question has haunted developers for years and still does today. When developing a distributed system, say a client-server-based system, how many servers do we need to be able to cope with all client requests? This is a pretty hard question to answer, it is like being a programmer and trying to decide whether the array you're declaring in your program is big enough. Since we cannot confidently answer either of these questions, we need to be open – think linked-lists in the context of the array problem.

In the context of distributed systems, the ability to add and remove resources, or in other words, the ability to be open, is paramount. The addition of resource is unproductive unless the resource can be used effectively. This means that the system needs to adapt to the existence of the new resources. Removing nodes is also related to adaptiveness to openness. Once a resource is removed the environment must be able to cope with the extra load. The information held by the resource that was removed may need to be moved to the remaining resource (assuming a scheduled failure).

3 Adaptiveness in Linda

The Linda coordination model has been quite successful in dealing with the intricacies of building ensembles of distributed components. Unfortunately none

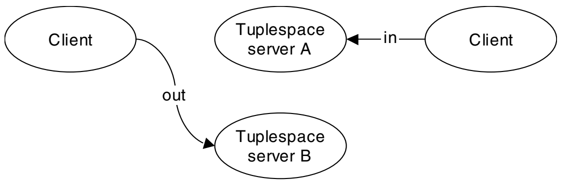


Fig. 2. Partitioned tuple spaces

of its extensions (ie. the Linda family of models) is able to cope with the demand for adaptiveness that exist in novel applications. In this section we will look at a few models based on Linda, namely MTS-Linda [12], PeerSpaces [4], Lime [22] and TOTA [15], to demonstrate how they cope with the various aspects of adaptiveness described in Section 2.

3.1 Tuple Distribution

Prior to getting into some of the models, is is wise to describe one of the main aspects of Linda models: tuple distribution. The way tuples are distributed differs from model to model. The literature describes a plethora of approaches for distributing tuple spaces. However, several major strategies can be identified: Centralization, Partitioning, Full Replication and Intermediate Replication.

- *Centralization* is a simple client-server distribution strategy where one specific server-machine operates the complete tuple space (as in TSpaces [28]). It can be accessed by clients that are arbitrarily located in a network (see Fig. 1).
  The centralized tuple-space server has the advantage of an easy implementation that basically attaches a network interface to an otherwise non-distributed tuple-space. However, it carries all the disadvantages of any centralization of services. The tuple-space server is most likely to become a bottleneck under high load induced from a large number of active clients, it is the single point of failure in the entire system, and it does not make a fair use of network resources.
- *Partitioning* of tuple spaces is a strategy in which tuples with common characteristics are co-located in one of a set of tuple-space servers (see Fig. 2). Requests with the same characteristics are then routed towards that machine. While simple partitioning (eg. co-locating tuples with same parity) might lead to unbalanced partitions, a carefully chosen hashing function on tuples can do better [2].
  Partitioning has the advantage of providing a distributed management of a tuple space including concurrent execution of operations on the partitions and thus slightly relaxes the problems of centralization. However it does include a centralization for certain sets of tuples. Also, the partitioning scheme handles reconfigurations very poorly. An automatic adaption of any hashing function and the induced distributed reorganization of the tuple-spaces content is complex and costly.

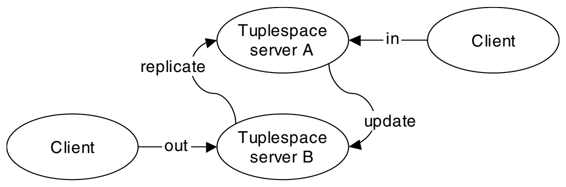


Fig. 3. Full replication

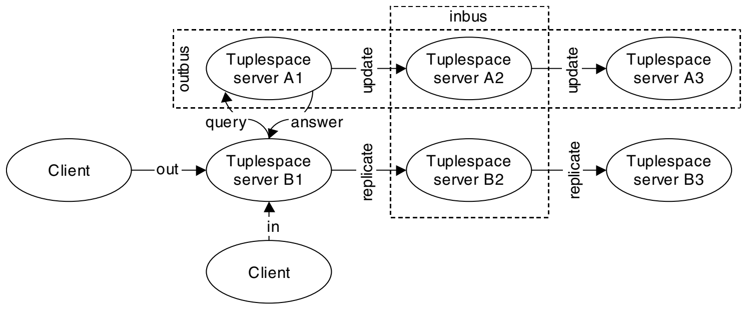


Fig. 4. Intermediate replication

– *Full replication* places complete copies of tuple spaces on several machines at different locations. Any addition and removal of data has to be replicated at all nodes so that searches for data can be performed locally on one machine (see Fig. 3).
  Full replication distributes the load for data-searches and inherently offers some support for fault-tolerance. But the cost of keeping the replicas consistent on all participating nodes is high and requires locking protocols or equivalents since multiple local searches may lead to the decision to remove the same piece of data [8].
– *Intermediate replication* has been proposed in the early implementations of Linda [7]. The schema uses a grid of nodes formed by logical intersecting "busses". Each node is part of exactly one *outbus* and one *inbus*. Data stored is replicated on all nodes of the outbus, whereas searches are performed on the inbus. As one inbus intersects all outbusses, it provides a complete view of the tuple spaces (see Fig. 4). With simulated nodes, the number of nodes can change dynamically while retaining the virtual grid [26].
  The intermediate replication schema allows for as many concurrent replications and searches for data as there are out- and inbusses respectively. The removal of some data, however, requires a consistent update on all nodes on the respective outbus.

Tuple distribution mechanisms as the ones described above play a role on the adaptiveness of Linda models. While none of these approaches seems to be useful (without modification) to the implementation of adaptable large-scale distributed Linda-like systems, the intermediate replication schema is the most general and flexible. It conceptually captures the centralized approach – a single outbus of size 1 – and the fully replication case – a single outbus of size n for n participating nodes.

Yet, none of the approaches described here scale well with the number of processes in the system and they cannot cope with dynamic aspects that are common place in these systems. Thus, long-term solutions for this problem are still due.

3.2 Linda-Based Models

Several Linda-based models have tried throughout the years to make the original Linda [11] more appropriate to deal with real-world applications. The first important modification of the original model occurred in 1989 when Gelernter realized that Linda lacked the ability to provide separation of concerns. In the original model all tuples were kept in the same (logical) location. The multiple tuple space (MTS) Linda [12] allowed the processes to create tuple spaces of their own and have the discretion of making them available to other processes.

MTS-Linda is far from being able to adapt to any change in the environment. Perhaps the reason is that conceptually MTS-Linda was a model for parallel computing where adaptiveness plays a much lesser role. Yet, it is worth pointing out that MTS-Linda was quickly recognized as an attractive paradigm for open distributed systems. We are a bit more precise with this statement: Linda *is* a good model for open distributed systems, but it only needs to worry more about real-world issues.

Melinda [13] was the first implementation provided by the Yale group for the MTS-Linda model that had a “flavor” of a distributed environment. Although it is unclear from Hupfer’s work what tuple distribution mechanism has been used in the implementation, Melinda is important in the context of adaptive Linda models because it assumes that processes are first-class objects that can be created, read, snapshot, destroyed, archived and re-activated. Such operations can be used to make the model more aware of certain adaptiveness aspects.

From the point of view of network unreliability there is nothing much Melinda can do. Melinda communication mechanism is standard and does not assume that messages can be re-routed or certain communication paths can be avoided based on the load/status of the network. Melinda is stronger in failure and mobility adaptiveness. Failures can be handled and the system can exhibit some level of adaptiveness *only if* it is a scheduled failure. In this case processes and tuple spaces (also first class objects) can be suspended and resumed in another location. The same scheme may be used to implement some level of mobility adaptiveness – migration of processes and data may also be implemented via a stop-resume approach.

Lime [22] was the first system which emphasized the dynamics of networked environment and tried to establish a coordination model in which it is accommodated in a natural way. In Lime, every agent can be mobile. It carries with it a local tuple space. There is no notion of a global tuple spaces, instead all local tuple spaces of agents co-located at some node are federated into a "*host-level transiently shared tuple space*". `in`-operations search this federated space for matches while `out`-operations drop a tuple into the tuple space that is bound to the agent. Lime agents therefore adapt to the mobility of agents and their current partial view of the conceptual global tuple space. It also breaks any static roles of components being either servers or clients to others.

Lime provides an additional operation, `out(l,t)`, to send a tuple `t` to some agent `l`. If that agent is co-located with the agent executing the primitive, the tuple is stored there. If not, it is stored with the current agent and transported with it until some co-location with `l` emerges. By that, the model does not prescribe any structure of distribution of tuples. They are somewhere and will eventually be at their destination.

PLinda [1] is a fault-tolerant implementation of the Linda model. Fault-tolerance is realized via the mechanisms of transactions. PLinda adds the operations `xstart` and `xcommit` which defines the start and end of transactions. A transaction can also end with the execution of a `xabort` which forces a rollback of all transactions operations. The users need to be able to identify transactions – the model leave the actual reaction to faults mostly to the user.

It is worth pointing out that PLinda is a closed system and that most of PLinda transactional semantics is not appropriate for open systems. There is a heavy locking mechanism implemented in PLinda that makes this approach almost impossible to be realized in a dynamic open distributed environment. This is true even though PLinda assumes that users have the discretion of identifying operations that are not required to have the implicit locks associated with them via the operation `nolock`.

Jeong *et Al.* [1] also discuss the ability that PLinda processes have of migrating from one machine to another. Migration is treated as a failure followed by a recovery on another node. This approach is consistent and is likely to work well in open environments if migration is not a frequent operation. The failure/recovery cost may be too high as it may have a domino effect to other processes.

PeerSpaces [4] is a proposal for a coordination model for peer-to-peer (P2P) networks. PeerSpaces premise is that P2P networks are very dynamic with regards to their topology for new peers can join and leave the network at any point. In other words, P2P networks are open in nature. Owning to this characteristic, P2P applications have to demonstrate a some degree of self-configuration. PeerSpaces aims at taking the burden of programming interaction between these peers away from the developers.

In order to achieve some degree of generality and thus be useful to a plethora of P2P applications, PeerSpaces allows the developers to define the structure of the data that will be stored in the coordination media. This is added to the ability that developers have to define an attribute for each individual datum.

The attributes are of three kind: *located* data are always fixed to one peer (local or remote); *generic* data are the closest to a standard Linda tuple (or datum) for the environment has discretion on where to place the data; *replica-*

ble data is used with data that is immutable and can safely (and cheaply) be replicated across the peers – its immutability does not require expensive consistency protocols to be implemented. The attributes are defined for each operation `write(d,t)`, where `t` is the attributed for the data. PeerSpaces does have some level of adaptiveness, particularly with regards to the *generic* attribute. The author indicate that these can migrate from peer to peer transparently at run-time if necessary (ie. due to load-balancing systems, etc.). The other two attributes are a mechanism that PeerSpaces provides to deal with special cases. For instance, replication of data may not be necessary if data is always maintained close to the requestors. For the same reason, it would be better if the developer did not have to think about *located* data at the time of design but rather let the environment decide – based on the application characteristics – if (and when) the data should be maintained fixed. The above means that PeerSpaces does not use the same distribution mechanism for the entire system. The closest classification would be that it uses full replication as well as partitioning.

Another interesting aspect of PeerSpaces is the definition of a *search space* for each data request. The space is based on the proximity from the origin of a request and works on a similar fashion as the TTL (time-to-live) in network packets. PeerSpaces adds the size of the search space, given by `h`, to the operations it defines: `take(d,h)` and `read(d,h)`. This mechanism allows PeerSpaces to discover data in newly added peers if they are within the defined search space. While PeerSpaces can deal with openness, nonetheless a mechanism in which the search space is not defined by the user could work better. The ideal search space could emerge from interactions between peers and converge to a nearly optimum to the particular application.

As for the other aspects of adaptiveness, namely mobility and network unpredictability, the authors quickly mention that possibility in PeerSpaces but do not elaborate in how this is achieved. In fact, it is unclear whether mobility would be achieved as part of the PeerSpaces system or via other facility implemented in the P2P network. The same is true for network unpredictability.

TOTA (Tuples on the Air) [15] is a take on the use of swarm abstractions in Linda which is complementary to what we propose in SwarmLinda (see Section 5). TOTA highlights the dynamic adaption of tuple flows with the applications structures. It emphasizes locality of perception of surrounding tuples as the context of agents and provides a complete abstraction from any underlying network structure. Tuples are propagated through a network of nodes with no fixed or predetermined location. Tuples have both a content and rules on their propagation which are used to scope tuples within the global space. Yet, tuples itself seem to be stationary.

At runtime, every node – which can also be mobile – maintains a list of neighbor node and provides an API to clients. The propagation rules are used to establish both structure in the overall network and routing of content contained in tuples towards some remote destination. A variety of overlay networks can be generated by that which dynamically adapt to the applications needs. With that, tuples can be coordinated to form swarms or flocks. At the base of that coordination is no global view but a completely decentralized local-sensing collection of equal peers in a network.

While TOTA seems to deal well with dynamic reconfiguration in the network, it is unclear whether the reconfiguration of overlay data structures (tuples and their replicas) may become an expensive operation if node movement is frequent. Another potential source of performance loss may be caused by the size of the lookup tables necessary to implement the propagation for a specific tuple – since tuples in TOTA are uniquely identified each one of them may end up having a entry in such a lookup table.

The programmability of propagation rules in TOTA, plus the algorithm used by the components to make efficient use of the propagation rule is what makes it very adaptable. The SwarmLinda approach described later in this paper assumes that developers are not responsible to define propagation rules neither need to be concerned about how to use the propagated information.

4 Swarming as Adaptive Mechanism

Swarm Intelligence [3, 14] is a new field of artificial intelligence that advocates that systems can be written in such a way that solutions *emerge* from local interactions between small agents. Locality and emergence are achieved via the use of stigmergy and swarming respectively.

In simplified terms, stigmergy is a form of communication that uses the environment as the main communication channel. Basically, local agents (ants, termites) can leave information on the environment that can be used by other agents in the system. This information is normally represented as a pheromone-like data.

Swarming is the collective activity resulting from all the agents using local information available in the environment to make decisions as well as updating local information as necessary, in a feedback loop. For instance, if ants leave trails to successful paths to food supplies, other ants can use the trails (marked in the environment) to decide whether to follow that scent or not. Negative feedback occurs because the pheromone that defines trails is volatile. Positive feedback occurs when an ant follows an already marked path and re-enforces it after it finds that it was indeed a good path. The swarming will make the system self-organize. That is, the system will reach a level of organization that was not directly programmed – it emerges from an open system of interacting components.

In the optimization area, the approach above has had some success because it avoids local optima. In general terms, this means that swarm systems converge but also make attempts to explore new (better) solutions. The exploration for new solutions occurs for the agents are not required to choose the most attractive choice. In fact, if they do that swarming would not take place. The choice of an agent is normally made stochastically. Below we have a typical probability formula as defined in [3]:

$$P_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in J_i^k} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}} \quad (1)$$

The formula (1) indicates that the probability of an agent to take a path (i, j) is given by a ratio of how attractive that path is over the summation of the attractiveness of all other paths. There are two levels of information being considered: a pheromone value (τ) and a distance to the next position (j), given by η. Negative and positive feedback is normally implemented using a formula as defined in below (from [3]):

$$\tau_{ij}(t) = (1 - \rho) \cdot \tau_{ij}(t-1) + \Delta\tau_{ij}(t) \quad (2)$$

The negative feedback occurs based on the value of ρ, where $0 \leq \rho \leq 1$. The pheromone value for the path (τ_{ij}) on the step $(t - 1)$ is decreased and then added to a re-enforcement value for that path ij given by $\Delta\tau_{ij}(t)$.

It turns out that stigmergy and swarming can be implemented in Linda to make the model inherit the adaptiveness characteristics of any swarm system. Generative communication as advocated can be seen as a stigmergic form of communication already. The swarming part can be done by abstracting Linda entities, in particular tuples and templates, as ants or food. Linda nodes can be seen as the environment in which information will be made available to the tuple- and template-ants. This abstraction is used in what we call SwarmLinda.

5 SwarmLinda

We have presented SwarmLinda in [27, 18] in condensed form while [19] is an extended description on which we build here. SwarmLinda uses several adaptations of algorithms taken from abstraction of natural multi-agent systems [3, 21].

5.1 Distribution Mechanism

The first area of SwarmLinda where abstraction from natural multi-agent systems can be used is in the distribution of tuples amongst the nodes. Historically, tuples have been distributed using various *static* mechanisms as described in Section 3.1. In SwarmLinda the partitioning of the tuple space is dynamic and based on the concept of brood sorting [21] used by ants.

Ants are able to sort different kinds of things they keep in the anthill such as food, larvae, eggs, etc. In an anthill these are normally sorted by their type. More importantly, ants do this process in spite of the amount of each type, thus being very scalable. The individuals that operate here are tuple-ants (as opposed to template-ants). The environment is the network of SwarmLinda nodes. The state is the set of tuples stored thus far.

A SwarmLinda implementation may use brood sorting as below in the process of tuple distribution. One could see tuples being grouped based on their template which will lead to the formation of clusters of tuples. In this process, tuples are the food and the ant is the active process representing the `out` primitive:

1. Upon the execution of an `out` primitive, start visiting the nodes.
2. Observe the kind of tuples (the template they match) the nodes are storing. Each out-ant should have a limited memory so it doesn't remember the information about the entire "network" of nodes but only the last few – this guarantees that the decision is based on local information.
3. Store the tuple in the node if nearby nodes store tuples matching the same template. The decision to store is made stochastically based on what has been defined in (1). This decision also considers a small random factor $[0, \xi]$ to enable tuples to be stored in other locations.
4. If nearby nodes do not contain similar tuples, randomly choose (using the random factor) whether to drop or continue to carry the tuple to another node.

In order to guarantee that the steps above work well, certain conditions must be satisfied. The out-ant should eventually be able to store the tuple. For each time the process decides *not* to store the tuple, the random factor will tend to ξ. This increases the chance of storing the tuple in the next step. Also the likelihood to store the tuple is also calculated stochastically based on the kinds of objects in memory – if most of the objects in memory are of the same kind as the one being carried out, the likelihood to store the tuple becomes very high.

The power of this approach can be compared with the common partitioning scheme. Partitioning is based primarily on the use of a hash function to decide where the tuple should be placed. Hashing is not able to cope with failures and changes in the application behavior. Failures in certain nodes may be fatal to the system while changes in application behavior may require changes in the hash function being used.

The approach described above is able to improve the availability of the system without having to count on costly techniques such a replication of data. In the ant-based approach, there are no assumptions about the behavior of applications, there is no pre-defined distribution schema, and there are no special scenarios implemented to deal with failures in a node.

5.2 Searching for Tuples

The distribution of tuples is only part of the problem – obviously these tuples need to be found. Ants look for food in the proximity of the anthill. Once found, the food is brought to the anthill and a trail is left so that other ants may know where food can be found. The ants know the way back to the anthill because they have a short memory of the last few steps they took and also because the anthill has a distinctive scent that can be tracked by the ants. In a tuple space context, one could view tuples as food. The locations where the tuples are stored can be seen as the terrain while the templates are seen as ants that wander in the locations in search of tuples. The anthill is the process that executed the operation.

The individuals are the template-ants, the environment consists of tuple-space nodes whose state is composed by the tuples stored and "scent" of different kinds of template that indicate a likelihood that matches for a template are available. The volatile scents disappear slowly over time. The tuple-searching ant should follow the following rules:

1. The first step is to diffuse the scent of the process in the node it is connected to and the node's neighborhood. This distinctive scent represents the anthill and will be tracked by the ants when they need to return to the anthill.
2. Check for a matching tuple at the current location. If a match is found, return to the origin location and leave scent for the template matched at each step ($\varDelta\tau_{ij}\left(t\right)$). They find their way back by using their short memory to direct them *towards* the anthill and by tracking the distinctive scent of the process (as described above). If no match is found, check the direct neighborhood.
3. If there are no scents around the current location that fit to the template, randomly choose a direction in the network of nodes to look for a tuple.
4. If there is a scent that indicates a direction for next step (matching scent), move one step towards that scent and start over. We not only want to guarantee adaptability in SwarmLinda, we also want to maintain the non-determinism when searching for tuples. We achieve this by adding a small random factor in a range of $[0, \xi]$ to each scent. This enables paths other than the one with the strongest scent to be chosen (as described in Section 4).
5. The life of an ant is limited to ensure that it does not seek for tuples that have not yet been produced. After each unsuccessful step without a match, the ant stops its search with a probability of γ. This factor is 0 in the beginning and increased by some $\varGamma$ with each unsuccessful step. $\varGamma$ itself also increases over time. When the ant decides to stop searching, it takes one of four actions:
   (a) Sleep for some time and then continue. This is a pure limitation of activity. If the ant has reached an area where no matching tuples have been produced for a long time, the ant will have a hard time to get out of that location. The sleeping would allow sometime for the system to change and maybe get to a state where tuples can be found in that location.
   (b) Die and be reborn after some time at the location the search started.
   (c) Materialize in some other random location and continue to search for tuples. This will perhaps lead the ant to a find a match but will not lead to an optimal trail from the original location to the tuple's location. However, the marked trail may be used by other template-ants that operate in that region and can help find optimal trails from their origins to tuples.
   (d) The template-ant simply stops – they become quiescent until a tuple-ant finds it. They can still move in their quiescent mode if forced by the environment, for example by production of tuples in nearby nodes. However, for all effects this template-ant is quiescent and the process that produced it is blocked.

   Which action is taken depends on the age of the ant. After an ant has slept several times, it then tries a rebirth. After some rebirths, it decides to rematerialize elsewhere. Finally it may decide to become quiescent. The last action (5d) is SwarmLinda's equivalent to blocking.

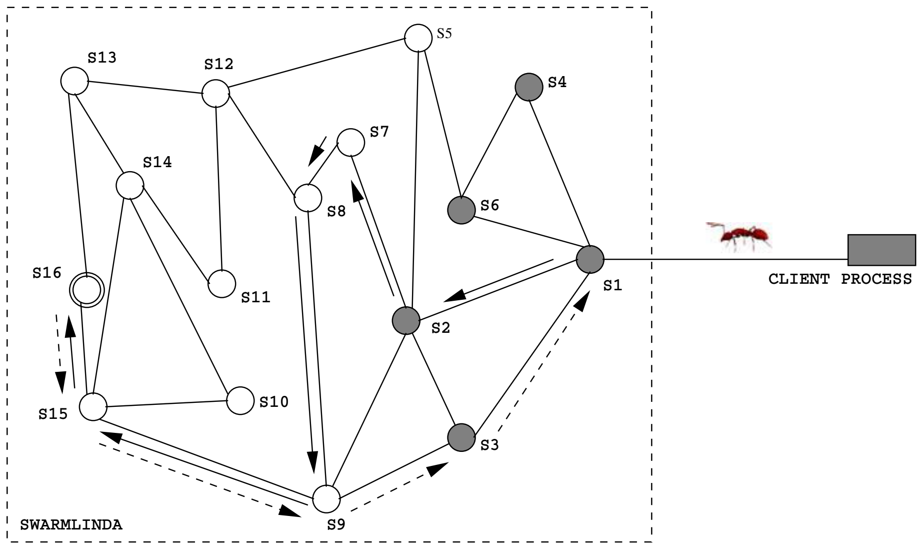


Fig. 5. Searching for a tuple

The result of the above is the emergence of application specific paths between tuple producers and consumers. Given that scents are volatile, the paths found can dynamically adapt to changes in the system – when consumers or producers join, leave or move within the system.

One question remains: how do ants find their way back to the anthill? We mentioned earlier that the anthill has a distinctive scent that can be tracked by the ants. In addition to this, ants have a short memory that allows them to follow back these steps *towards* the anthill.

It is worth noticing that ants do not tend to change their direction radically. Since they know where they came from (from their short memory) they avoid going back to that direction. This does not guarantee that an ant finds the anthill, but it maximizes its chances. The idea is that the last few steps in the ant's short memory are used to lead it to the anthill. Once moving in the *right* direction and the anthill is found the trail left by this ant can be used in future searches.

Fig. 5 shows an example of the search algorithm. A client connected to node S_1 diffuses its scent on nodes S_1, S_2, S_3, S_4, S_6. Later, a template-ant goes searching for a tuple. For the purposes of this example let us assume the template-ant can remember the last two steps it took. The template-ant wanders searching for a tuple making a decision at each node. After a few steps the template-ant finds a matching tuple in node S_{16} – the path it took until it found the tuple was [S_1, S_2, S_7, S_8, S_9, S_{15}, S_{16}]. After the tuple is found, the template-ant uses its short memory to return to the anthill. The first two steps returning are taken based on what is in memory: [S_{15}, S_9]. Next, the template-ant tries to track the scent of the anthill. In S_9 the template-ant is influenced by such a scent and moves to S_3, S_1, and finally back to the client. Observe that the return path is not necessarily the path it took to find the tuple. In the end, the return path [S_{16}, S_{15}, S_9, S_3, S_1] was marked with the scent for that particular template.

The returning ant works similarly to a backward ant in AntNet [6]. The returning ant must apply the positive feedback to the successful path as indicated in (2). The negative feedback part is done by evaporation and is controlled by the SwarmLinda system.

Compare this approach with one standard mechanism to find distributed tuples: hashing. Tuples are normally searched based on a hash function that takes the template as the input and generate a location where the tuple can be found as the output. Hashing is definitely fast but unfortunately not very adaptive. The determinism that exist in hash functions forces tuples with the same template to *always* be placed in the same location no matter the size of the system, thus causing bottleneck problems if tuples matching such template are in demand in the system.

Although in a SwarmLinda tuples matching the same template would *tend* to stay together, this is not necessarily true in all cases. If such tuples are being produced in locations far enough from each other the tuples will remain separate and create clusters across all the system. This should avoid the creation of bottlenecks when tuples of a certain template are required by many processes. As searches start from various locations, tuples tend to be retrieved from the closest cluster from the source of the search.

Another problem with hashing approaches is that they are not fault tolerant – if a tuple is being hashed to a location, that location is expected to be working. Hashing may be made fault-tolerant (based on conflict resolutions techniques) but its implementation is normally cumbersome, its use in practice may be very expensive, and its effectiveness doubtful. From the point of view of swarm techniques, failures are just another change in the environment. Failures would behave like ants trying to search for food in a food supply that was suddenly destroyed. Surely this is not a problem and the ants will only starve if food cannot be found elsewhere.

5.3 Dealing with Openness

Openness is known to be one of the main challenges in distributed systems – the ability of a system to deal with changes can be a great asset. For instance, in open Linda systems the need for tuples of specific formats (templates) may change overtime.

In order to enable a SwarmLinda to show an adaptive behavior for collections of similar tuples, we again use tuple-ants as the individuals. The environment is again a terrain of nodes that has scents as the state.

In SwarmLinda, we want tuples matching the same template to be kept together (as described in Section 5.1) but we do not want them to be fixed to a given location. Instead, we want them to dynamically adapt to changes.

Given a function $Sc : T \rightarrow S$ on templates and tuples and a relation $C : S \times S$ on scent that defines similarity of scent. One can say that if the template te and tuple tu match, then $Sc(te), Sc(tu) \in C$.

1. A new tuple-ant that carries a tuple tu emits $Sc(tu)$ at its origin. A new template-ant that carries a template te emits $Sc(te)$ at its origin.
2. Template-ants remain at that position and never move.
3. Tuple-ants sense their environment for a scent similar – as given by C – to $Sc(tu)$. If there is such, then other template- or tuple-ants are around.
4. Based on the strength of the detected scent plus the small random factor $[0, \xi]$, the tuple-ant decides to move towards that direction or to stay where it is.

The above causes tuples to stay closer to where other similar tuples are needed or are being produced (based on the number of `in` and `out` primitives executed) even if this consists of migrating from one node to another. This would also have an effect on the distribution mechanism explained in Section 5.1. When a tuple is being stored the scent left by previous `in` and `out` primitives should also be considered when deciding to drop the tuple in the current node or to keep "walking" through the terrain of nodes searching for a good place to drop the tuple.

Swarming also allows SwarmLinda to be open in terms of configuration of the nodes. It is not uncommon that systems become overloaded due to the inability to predict demand for performance in the future. In SwarmLinda, new nodes can be added as necessary – the new nodes will be explored by the ants as long as they are part of at least one current nodes neighborhood. Therefore the addition of a node has to ensure that the new node is added to some other node's neighbor list. This aspect of openness is not as easy to guarantee in other implementations because the topology is normally implemented in the Linda system itself.

5.4 Balancing Tuple- and Template Movement

In the preceding algorithms, we always identified either the tuple-ants or the template-ants as individuals that move and perform a continued search. In this section we describe an intermediate approach where ants can be both tuples and templates. Every tuple- and template-ant decides after its birth whether it goes out to other nodes seeking matches or stays at its origin until it is found by some other ant.

Consider an application where one node consumes a lot of tuples that are generated on other nodes. If trails from the producers to the consumer are found – and these can be found by programming a tuple-ant with the algorithm from Section 5.2 – it makes no sense to have the consumer start template-ants that seek the producers. Based on the system history (of scents) it is known where a consumer is and what the path is, so tuple-ants should be pushed there while the template-ants at the consumer should remain stationary and basically wait for a matching tuple-ant to appear. But if the former consumer starts to be a producer after the calculation of some results, it might become reasonable to start template-ants from the former producers to reach out for the result.

Our algorithm should lead to a dynamic balance between active and passive ants that takes into account the current producer/consumer configuration in the system.

For the algorithm, the individuals are tuple- and template-ants. The environment is still the terrain of nodes. The state at each location includes two scents: One scent indicates whether the location is visited successfully by other ants – it is an attraction – or not – it is an outsider. Success means that the visiting ant found a match at this location. We call this the visitor scent. The second scent, the producer-consumer scent ranges over $[-\phi, \phi]$. Positive values indicate that the matches that took place were such that a visiting template-ant retrieved a tuple from that location – showing that the location is a producer of information. A negative scent indicates that visiting tuple-ants were matched with a template at that location – the location is a consumer of tuples.

Tuple- and template-ants follow the algorithms from Section 5.2 to find matching templates resp. tuples. If a tuple-ant finds a match, it neutralizes a bit of producer-consumer scent at the location. When a template-ant finds a match, it adds a bit of this scent at the location. Both kinds of ants leave a bit of visitor scent in the case of success.

When a new ant is born, it will either be a tuple- or a template-ant depending on the kind of operation requested. A new tuple-ant emits a bit of producer-consumer scent at the location of its birth, a template-ant neutralizes some.

These ants can behave in two different ways: Either they are active and move around following the search algorithms as described above, or they are passive and remain at their origin to be found by others.

The further fate of a new ant depends on the current state of the location where they are born. This state distinguishes producing and consuming locations and whether the location is attractive for visitors. The following table shows how new ants behave based on these two characteristics:

	Producer	Consumer
Attraction	Passive tuple-ant Passive/active template-ant	Active/passive tuple-ant Passive template-ant
Outsider	Active tuple-ant Passive template-ant	Passive tuple-ant Active template-ant

If a producer is visited by many ants, there is no need to send out tuple-ants. Template-ants can be passive or active depending on how many visitors satisfy them – it is important to keep a balance between active and passive template-ants. The ratio of passive/active can be adjusted. For an attractive consumer, template-ants may remain passive. Owing to the same global-balance argument above, tuple-ants can also be active. If a producer is not visited by many other ants, it will send out its tuples-ants to find matches. Its template-ants can remain passive.

If a consumer is not visited by many other ants, it will send out active template-ants to find matches and generate passive tuple-ants to attract other

	Melinda	Lime	PLinda	PeerSpaces	TOTA	SwarmLinda
Network Unpredictability	×	√	×	○	√	√
Failures	○	○	√	√	√	√
Mobility (**p**hysical)/(**d**ata)	○	√ (p/d)	○(d)	○	√ (p/d)	√ (p/d)
Openness	×	○	×	√	√	√

Fig. 6. Summary of how some models deal with aspects of adaptiveness

active template-ants. The generation of passive tuple-ants improves its chances of becoming an attraction.

The algorithm can be compared to the intermediate replication scheme in Linda systems. There, an `in` leads to a broadcast of the template on the inbus and to a search for a matching tuple. An `out` leads to the replication of the tuple on the outbus where the local lists of waiting tuples are then searched. This seems to resemble the idea of having tuple-ants and template-ants go out and seek for matches. However, the inbusses and outbusses in intermediate replication are usually very static. In the SwarmLinda algorithm the balance between tuple- and template-ants is highly dynamic and adapts to the current behavior of the running applications.

Overall, the search algorithms presented in this section will eventually find matching tuples or templates. This is a consequence of using a random walk: after a finite number of steps, all possible nodes will have been visited. This is similar to a Linda implementation where a predefined list of nodes is searched – in the end all nodes that can possibly have a match have been visited.

6 Conclusion

This paper has discussed several aspects of adaptiveness and their importance to large scale dynamic systems. The paper focused on investigating how the aspects related to adaptiveness to network unpredictability, failures, mobility and openness have been tackled by various Linda systems. Figure 6 summarizes our discussion.

In Figure 6, a × indicates that the model does not demonstrate that it can handle that particular aspect of adaptiveness. A √ indicates that the model does present levels of that particular aspect. And a ○ indicates that (maybe indirectly) the model does present some *potential* in dealing with that particular aspect.

From Figure 6 one can clearly see that TOTA and SwarmLinda are very adaptable. The difference between these systems lie on the fact that SwarmLinda wants designers to be oblivious to the swarming that may happen in the system. Ideally, a SwarmLinda designer would not need to have any understanding of swarming and related concepts such as the propagation rules defined in TOTA.

We have demonstrated how the various aspects of adaptiveness can be uniformly dealt with in SwarmLinda. It does not make any distinction between the aspects of adaptiveness, tackling them in the same way via the mechanism of swarming and stigmergic communication. A SwarmLinda implementation is being developed and will be made available in the website [17].

References

[1] Brian G. Anderson and Dennis Shasha. Persistent linda: Linda + transactions + query processing. In J.P. Banâtre and D. Le Métayer, editors, *Research Directions in High-Level Parallel Programming Languages*, number 574 in LNCS, pages 93–109. Springer, 1991. 213, 215, 220

[2] Robert Bjornson. *Linda on Distributed Memory Multiprocessors*. PhD thesis, Yale University Department of Computer Science, 1992. Technical Report 931. 217

[3] E. Bonebeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford Press, 1999. 222, 223

[4] N. Busi, A. Montresor, and G. Zavattaro. Data-driven coordination in peer-to-peer information systems. *International Journal of Cooperative Information Systems*, 2004. to appear. 213, 217, 220

[5] Luca Cardelli. Wide area computation. *Lecture Notes in Computer Science*, 1644:10pp, 1999. 213

[6] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998. 227

[7] Nicholas Carriero and David Gelernter. The S/Net's linda kernel. *ACM Transactions on Computer Systems*, 4(2):110–129, 1986. 218

[8] A. Corradi, L. Leonardi, and F. Zambonelli. Strategies and protocols for highly parallel linda servers. *Software: Practice and Experience*, 28(14), 1998. 218

[9] Rocco de Nicola, Gian Luigi Ferrari, and R. Pugliese. Klaim: a kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering (Special Issue on Mobility and Network Aware Computing)*, 1998. 213

[10] Keith Decker, Karia Sycara, and Mike Williamson. Intelligent adaptive information agents. In Ibrahim Imam, editor, *Working Notes of the AAAI-96 Workshop on Intelligent Adaptive Agents*, Portland, OR, 1996. 214

[11] David Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985. 212, 219

[12] David Gelernter. Multiple tuple spaces in Linda. In E. Odijk, M. Rem, and J.-C. Syre, editors, *PARLE '89, Vol. II: Parallel Languages*, LNCS 366, pages 20–27, 1989. 213, 217, 219

[13] Susanne Hupfer. *MELINDA: LINDA with Multiple Tuple Spaces*. PhD thesis, Yale University, February 1990. YALE/DCS/RR-766. 219

[14] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001. 222

[15] Marco Mamei, Franco Zambonelli, and Letizia Leonardi. Tuples on the air: a middleware for context-aware computing in dynamic networks. In *Proc. of the 2nd International Workshop on Mobile Computing Middleware at the 23rd International Conference on Distributed Computing Systems (ICDCS)*, pages 342–347, Providence, RI, USA, May 2003. IEEE. 213, 217, 221

[16] Ronaldo Menezes, Andrea Omicini, and Mirko Viroli. On the semantics of coordination models for distributed systems: The logop case study. In *Proc. of 2nd International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA 2003)*, 2003. to appear in the Electronic Nodes in Theoretical Computer Science. 213

[17] Ronaldo Menezes and Robert Tolksdorf. Swarmlinda website. `http://cs.fit.edu/~rmenezes/SwarmLinda`. 230

[18] Ronaldo Menezes and Robert Tolksdorf. A new approach to scalable linda-systems based on swarms. In *Proceedings of ACM SAC 2003*, pages 375–379, 2003. 223

[19] Ronaldo Menezes and Robert Tolksdorf. A new approach to scalable linda-systems based on swarms (extended version). Technical Report CS-2003-04, Florida Institute of Technology, Department of Computer Sciences, 2003. `http://www.cs.fit.edu/~tr/cs-2003-04.pdf`. 223

[20] Andrea Omicini and Franco Zambonelli. TuCSoN: a coordination model for mobile information agents. In David G. Schwartz, Monica Divitini, and Terje Brasethvik, editors, *1st International Workshop on Innovative Internet Information Systems (IIIS'98)*, pages 177–187, Pisa, Italy, 8-9 June 1998. IDI – NTNU, Trondheim (Norway). 213

[21] H.V.D. Parunak. "go to the ant": Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997. 223

[22] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. LIME: Linda meets mobility. In *International Conference on Software Engineering*, pages 368–377, 1999. 213, 217, 219

[23] Antony Rowstron. Mobile co-ordination: Providing fault tolerance in tuple space based co-ordination languages. In P. Ciancarini and P. Wolf, editors, *Coordination Languages and Models (Coordination'99)*, number 1594 in LNCS, pages 196–210. Springer Verlag, 1999. 215

[24] Jim Snyder and Ronaldo Menezes. Using logical operators as an extended co-ordination mechanism in linda. In *Coordination Models and Languages*, pages 317–331, 2002. 213

[25] R. Tolksdorf and A. Rowstron. Evaluating fault tolerance methods for large-scale linda-like systems. In *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, 2000. 215

[26] Robert Tolksdorf. Laura - a service-based coordination language. *Science of Computer Programming, Special issue on Coordination Models, Languages, and Applications*, 1998. 213, 218

[27] Robert Tolksdorf and Ronaldo Menezes. Using swarm intelligence in linda systems. In Andrea Omicini, Paolo Petta, and Jeremy Pitt, editors, *Proceedings of the Fourth International Workshop Engineering Societies in the Agents World ESAW'03*, 2003. To appear. 213, 223

[28] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T spaces. *IBM Systems Journal*, 37(3):454–474, 1998. 217

Self-Organization in Multi Agent Systems: A Middleware Approach

Marco Mamei and Franco Zambonelli

Dipartimento di Scienze e Metodi dell'Ingegneria – Università di Modena e Reggio Emilia
Via Allegri 13 – Reggio Emilia – ITALY
{mamei.marco,franco.zambonelli}@unimo.it

Abstract. Self-organization in multi agent systems requires two main building blocks: adaptive and uncoupled interaction mechanisms and context-awareness. Here we show how the middleware TOTA (Tuples On The Air) supports self-organization by providing effective abstractions for the above two building-blocks. TOTA relies on spatially distributed tuples for both supporting adaptive and uncoupled interactions between agents, and context-awareness. Agents can inject these tuples in the network, to make available some kind of contextual information and to interact with other agents. Tuples are propagated by the middleware, on the basis of application specific patterns, defining sorts of "computational fields", and their intended shape is maintained despite network dynamics, such as topological reconfigurations. Agents can locally "sense" these fields and rely on them for both acquiring contextual information and carrying on distributed self-organizing coordination activities. Several application examples in different scenarios show the effectiveness of our approach.

1 Introduction

IT scenarios, at all levels, are witnessing a radical change: from systems constituted by components fixedly and strictly coupled at design time, to systems based on autonomous, uncoupled and transiently interacting components [31]. In computer science, applications have always been built by adopting programming paradigms rooted on strictly coupled, non-autonomous components. Following this approach, components are coupled at design time by fixed interaction patterns. Although simple, this approach turned out to be really brittle and fragile, not being able to cope with reconfiguration and faults. Only in recent years, the research on software agents has fostered new programming paradigms based on autonomous components (i.e. components with a separated thread of execution and control) interacting to realize an application [6, 10, 21].

This shift of paradigm is well motivated by the robustness, scalability and flexibility of systems based on these autonomous components: if a component breaks down, the others can re-organize their interaction patterns to account for such a failure, if new components are added to the system, they can discover which other components are present and start to interact with them. The key element leading to such robust, scalable and flexible behaviors is self-organization. Autonomous

© Springer-Verlag Berlin Heidelberg 2004

components must be able to self-organize their activity patterns to achieve goals, that possibly exceed their capabilities as single individuals, despite, and possibly taking advantage, of environment dynamics and unexpected situations [20, 18, 30]. Nature, for example, "adopts" these ideas at all scales (e.g. in social insects like ants, in cells like in the immune system or neurons in the brain) [4, 5, 7].

The first signs of this shift of paradigm can be found in modern distributed computing where the inherent dynamism of networks (e.g. delays and link unavailability) forces distributed application components to autonomously adapt and self-organize their behavior to such dynamism. On the one hand, Internet applications, traditionally built following a client-server approach, are gradually replaced by their peer-to-peer (P2P) counterpart. By investing on peers autonomy, P2P applications can self-organize their activities to achieve unprecedented levels of robustness and flexibility (e.g. peers can dynamically discover communication partners and autonomously engage, also third-party, interaction patterns). On the other hand, components' autonomy and self-organization is at the basis of ubiquitous and pervasive computing, where intrinsic mobile components are connected in wireless, amorphous networks. In such a dynamic scenario, in fact, agents have to constantly rearrange and self-organize their activities to take into account the ever changing environment.

Unfortunately, we still do not know how to program and manage these kind of autonomous self-organizing systems. The main conceptual difficulty is that we have direct control only on the agents' local activities, while the application task is often expressed at the global scale [5, 7]. Bridging the gap between local and global activities is not easy, but it is possible: distributed algorithms for autonomous sensor networks have been proposed and successfully verified, routing protocols is MANET (in which devices coordinate to let packets flow from sources to destinations) have been already widely used. The problem is still that the above successful approaches are ad-hoc to a specific application domain and it is very difficult to generalize them to other scenarios. There is a great need for general, widely applicable engineering methodologies, middleware and APIs to embed, support and control self-organization in multiagent systems [1, 13, 26, 31]. From our point of view, self-organization is a sort of distributed coordination and its main building blocks are those constituting the core of agents' autonomy: *(i)* adaptive and uncoupled interaction mechanisms and *(ii)* context-awareness (i.e. the fundamental capability for an agent to be aware of its operational environment). With regard to the first point, environment dynamism and transiently connected components call for flexible, adaptive and uncoupled interactions. Moreover, by its very nature, coordination requires context-awareness. In fact, an agent can coordinate with other agents only if it is somehow aware of "what is around" i.e. its context. However, when agents are embedded in a possibly unknown, open and dynamic environment (as it is in the case of most pervasive computing scenarios), they can hardly be provided with enough *a priori* up-to-date contextual knowledge. Starting from these considerations, it is fundamental to provide agents with simple, easy to be obtained, and effective contextual information, supporting and facilitating their coordination activities in a robust and adaptive way.

The contribution of this paper is to show how the abstractions promoted by a novel middleware infrastructure called TOTA ("Tuples On The Air"), suit the need of self-organization. Coherently with the above considerations, the key objective of TOTA is to define a single abstraction both to: *(i)* promote uncoupled and adaptive interactions; and *(ii)* provide agents with simple, yet expressive, contextual information to actively support adaptivity, by discharging application components from dealing with network and application dynamics. To this end, TOTA relies on spatially distributed tuples, to be injected in the network and propagated accordingly to application-specific patterns. On the one hand, tuple propagation patterns are dynamically re-shaped by the TOTA middleware to implicitly reflect network and applications dynamics, as well as to reflect the evolution of coordination activities. On the other hand, application agents have simply to locally "sense" tuples to acquire contextual information, to exchange information with each other, and to implicitly and adaptively orchestrate their coordination activities. To take a metaphor, we can imagine that TOTA propagates tuples in the same way as the laws of nature provides propagating fields in the physical space: although particles do not directly interact with each other and can only locally perceive such fields, they exhibit globally orchestrated and adaptive motion patterns.

This paper is organized as follows. Section 2 overviews the TOTA approach and its implementation. Section 3 describes some application examples, showing how can TOTA be effectively applied to different scenarios. Section 4 present performance and experiments. Section 5 discusses related works. Section 6 concludes and outlines future works.

2 Tuples on the Air

The driving objective of our approach is to address together the two requirements introduced in the previous section (uncoupled and adaptive interactions and context-awareness), by exploiting a unified and flexible mechanism to deal with both context representation and agents' interactions.

In TOTA, we propose relying on distributed tuples for both representing contextual information and enabling uncoupled interaction among distributed application components. Unlike traditional shared data space models [12], tuples are not associated to a specific node (or to a specific data space) of the network. Instead, tuples are injected in the network and can autonomously propagate and diffuse in the network according to a specified pattern. Thus, TOTA tuples form a sort of spatially distributed data structure able to express not only messages to be transmitted between application components but, more generally, some contextual information on the distributed environment.

To support this idea, TOTA is composed of a peer-to-peer network of possibly mobile nodes, each running a local version of the TOTA middleware. Each TOTA node holds references to a limited set of neighboring nodes. The structure of the network, as determined by the neighborhood relations, is automatically maintained and updated by the nodes to support dynamic changes, whether due to nodes' mobility or to nodes' failures. The specific nature of the network scenario determines how each node can find its neighbors: e.g., in a MANET scenario, TOTA nodes are found within the range of their wireless connection; in the Internet they can be found via an expanding ring search (the same used in most Internet peer-to-peer systems [24]).

Upon the distributed space identified by the dynamic network of TOTA nodes, each component is capable of locally storing tuples and letting them diffuse through the network. Tuples are injected in the system from a particular node, and spread hop-by-hop according to their propagation rule. In fact, a TOTA tuple is defined in terms of a "content", and a "propagation rule".

$$\boxed{T=(C, P)}$$

The content C is an ordered set of typed fields representing the information carried by the tuple. The propagation rule P determines how the tuple should be distributed and propagated in the network. This includes determining the "scope" of the tuple (i.e. the distance at which such tuple should be propagated and possibly the spatial direction of propagation) and how such propagation can be affected by the presence or the absence of other tuples in the system. In addition, the propagation rules can determine how tuple's content should change while it is propagated. In fact, tuples are not necessarily distributed replicas: by assuming different values in different nodes, tuples can be effectively used to build a distributed overlay data structure expressing some kind of contextual and spatial information. So, unlike traditional event based models, propagation of tuples is not driven by a publish-subscribe schema, but it is directly encoded in tuples' propagation rule and, unlike an event, can change its content during propagation (see Figure 1).

The spatial structures induced by tuples propagation must be maintained coherent despite network dynamism. To this end, the TOTA middleware supports tuple propagation actively and adaptively: by constantly monitoring the network local topology and the entry of new tuples, the middleware automatically re-propagates tuples as soon as appropriate conditions occur. For instance, when new nodes get in touch with a network, TOTA automatically checks the propagation rules of the already stored tuples and eventually propagates the tuples to the new nodes. Similarly, when the topology changes due to node movements, the distributed tuple structure automatically changes to reflect the new topology. For instance, Figures 2 shows how the structure of a distributed tuple can be kept coherent by TOTA in a MANET scenario, despite dynamic network reconfigurations.

From the application components' point of view, executing and interacting basically reduces to inject tuples, perceive local tuples and local events, and act accordingly to some application-specific policy. Software components on a TOTA node can inject new tuples in the network, defining their content and their propagation rule. They have full access to the local content of the middleware (i.e., of the local tuple space), and can query the local tuple and one-hop neighbors tuple spaces – via a pattern-matching mechanism – to check for the local presence of specific tuples. In addition, components can be notified of locally occurring events (i.e., changes in tuple space content and in the structure of the network neighborhood).

The overall resulting scenario is that of applications whose agents: *(i)* can influence the TOTA space by propagating application-specific tuples; *(ii)* execute by being influenced in both their internal and coordination activities by the locally sensed tuples; and *(iii)* implicitly tune their activities to reflect network dynamics, as enabled by the automatic re-shaping of tuples' distributions of the TOTA middleware.

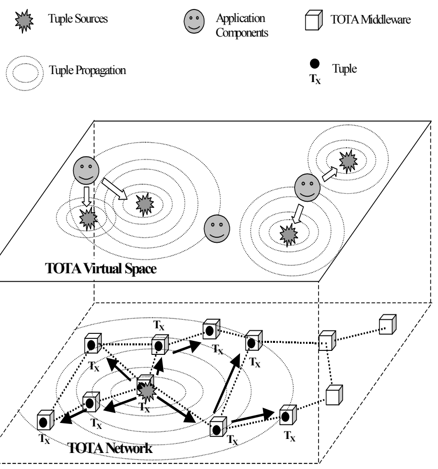


Fig. 1. The General Scenario of TOTA: application components live in an environment in which they can inject tuples that autonomously propagate and sense tuples present in their local neighborhood. The environment is realized by means of a peer-to-peer network in which tuples propagate by means of a multi-hop mechanism

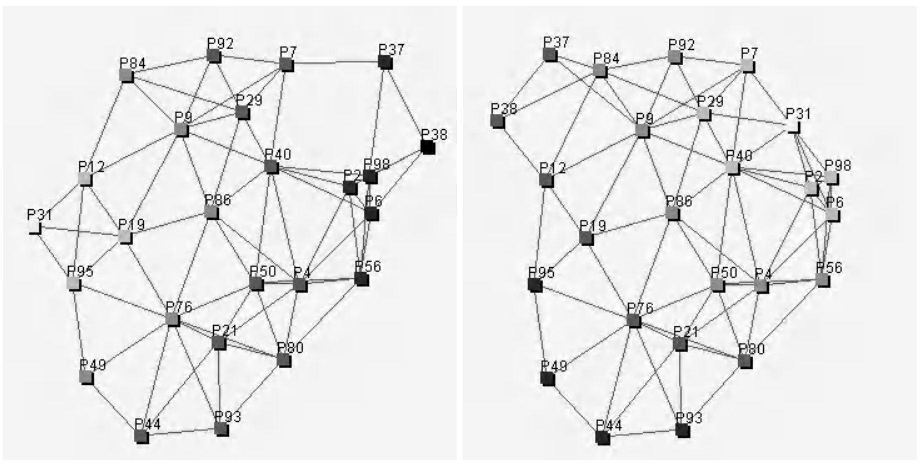


Fig. 2. (left) P31 propagates a tuple that increases its value by one at every hop. Tuple hop-value is represented by node darkness. (right) When the tuple source P31 or other nodes move, all tuples are updated to take into account the new topology

2.1 Implementation

From an implementation point of view, we developed a first prototype of TOTA running on Laptops and on Compaq IPAQs equipped with Linux, IEEE 802.11b WLAN cards, and Java (J2ME-CDC, Personal Profile). Devices connect locally in the MANET mode (i.e. without requiring access points) creating the skeleton of the TOTA network. Tuples are being propagated through multicast sockets to all the nodes in the one-hop neighbor. The use of multicast sockets has been chosen to improve the communication speed by avoiding 802.11b unicast handshake (please note that multicast in one-hop radius is assumed to be provided in these networks). By considering the way in which tuples are propagated, TOTA is very well suited for this kind of broadcast communication. We think that this is a very important feature, because it will allow in the future implementing TOTA also on really simple devices [14] that cannot be provided with sophisticate communication mechanisms. Other than this communication mechanism, at the core of the TOTA middleware there is a simple event-based engine, that monitors network reconfigurations and the entry of new tuples and react either by triggering propagation of already stored tuples or by generating an event directed to the event interface.

Since the effective testing of TOTA would require a very large number of devices, we have implemented an emulator to analyze TOTA behavior in presence of hundreds of nodes. The emulator, developed in Java, enables examining TOTA behavior in a MANET scenario, in which nodes topology can be rearranged dynamically either by a drag and drop user interface or by autonomous nodes' movements. More details on the TOTA architecture can be found in [16], more details on the TOTA programming model can be found in [17]. Finally, a simplified applet version of the TOTA emulator can be accessed and used at our research group Web site (http://www.agentgroup.unimo.it).

3 Application Examples

In this section, to prove the generality of our approach, we will show how to exploit TOTA to solve several problems typical of dynamic network scenarios, by simply implementing different tuples' propagation rules.

3.1 Motion Coordination

To show the capability of achieving globally coordinated behaviors with TOTA, we focus on a specific instance of the general problem of motion coordination. Motion coordination has been widely studied in several research areas: robotics, simulations, pervasive computing, multi agent systems, etc. Among the others, a particularly interesting and successful approach is the one that exploit the idea of potential fields to direct the movement of the involved entities [15, 27]. As a first example, we will consider the problem of letting a group of mobile components (e.g., users with a PDA or robots) move maintaining a specified distance from each other. To this end, we can take inspiration from the mechanism used by birds to flock [7]: flocks of birds stay together, coordinate turns, and avoid each other, by following a very simple swarm algorithm. Their coordinated behavior can be explained by assuming that each bird tries to maintain a specified separation from the nearest birds and to match nearby birds' velocity. To implement such a coordinated behavior in TOTA, each component can generate a tuple *T=(C,P)* with following characteristics:

C = (FLOCK, nodeName,val)
P = ("val" is initialized at 2, propagate to all the nodes decreasing by one in the first two hops, then increasing "val" by one for all the further hops)

Thus creating a distributed data structure in which the *val* field assumes the minimal value at specific distance from the source (e.g., 2 hops). This distance expresses the intended spatial separation between components in the flock. To coordinate movements, components have simply to locally perceive the generated tuples, and to follow downhill the gradient of the *val* fields. The result is a globally coordinated movement in which components maintain an almost regular grid formation by clustering in each other *val* fields' minima.

To test the above coordination mechanism we used the emulator: the snap-shots of Figure 3 shows a MANET scenario in which a group of four components (in black) proceeds in a flock, maintaining a one hop distance. The other nodes in the network remain still and just store and forward flocking tuples.

Another interesting example of motion coordination, is the problem of letting mobile users to meet somewhere. Here we can imagine that each member of the meeting group injects a tuple with the following characteristics:

C = (MEET, nodeName, val)
P = ("val" is initialized at 0, propagate to all the nodes increasing "val" by one for all the further hops)

By relying on this tuple, we can realize different meeting policies:

1. A group of users wants to meet in the point where member *x* is located. This is the simplest case and each user can move by following downhill the tuple having in its content *x* as *nodeName*. It is interesting to notice that this approach works even if person *x* moves after the meeting has been scheduled. The meeting will be automatically rescheduled in the new minimum of *x*'s tuple.
2. A group of users wants to meet in the point that is between them (their "barycenter"). To this purpose each user *i* can follow downhill a linear combination of all the other MEET tuples. In this way all the users "fall" towards each other, and they meet in the point that is in the middle. It is interesting to notice, that this "middle point" is evaluated dynamically and the process takes into consideration the crowd or unexpected situations. So if some users encounter a crowd in their path, the meeting point is automatically changed to one closer to these unlucky users.

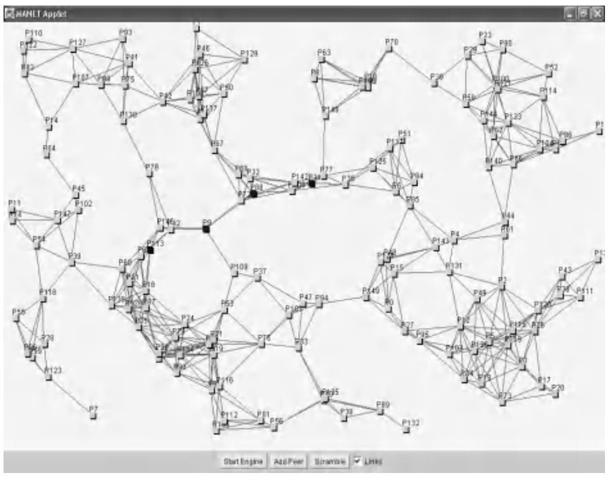


Fig. 3. Flocking in the TOTA Emulator. Cubes are the nodes of a mobile ad-hoc network, with arcs connecting nodes in range with each other. Black cubes are involved in flocking, moving by preserving a 2 hop distance from each other

3.2 Modular Robot Control

Another interesting application scenario, is in the control of a modular robot [29]: a collection of simple autonomous actuators with few degrees of freedom connected with each other. A distributed control algorithm is executed by all the actuators that coordinate to let the robot assume a global coherent shape or a global coherent motion pattern (i.e. gait). Currently proposed approaches [27] adopts the biologically inspired idea of hormones to control such a robot. Hormone signals are similar to content based messages, but have also the following unique properties: they propagate through the network without specific destinations, their content can be modified during propagation and they may trigger different actions for different receivers. The analogies between hormones and TOTA tuples are evident and, in fact, we were able to easily implement a similar control algorithm on top of TOTA. The algorithm has been tested on the 3D modular robot simulator available at [22]. Following the approach proposed in [27], we will consider the implementation of a caterpillar gait on a chain-typed modular robot, composed by actuators having a single motorized degree of freedom (see figure 4-right). Each robot node (i.e. actuator) will be provided with a TOTA middleware, and with an agent driving its motor. In particular the head agent, the tail agent and the body agents will drive the head module, the tail module and the body modules respectively.

The head agent starts the movement by injecting a *caterpillar-tuple*. The tail agent injects the *gait-tuple*, upon the receipt of a new *caterpillar-tuple*.

The gait-tuple is simply a tuple notifying that the gait has been completed, it simply propagates from the tail to the head (i.e. it has a broadcast propagation rule) without storing. The caterpillar-tuple has the following structure:

Current state	*New state*	*New angle*
INIT	*A*	*+45 deg*
A	*B*	*+45 deg*
B	*C*	*-45 deg*
C	*D*	*-45 deg*
D	*A*	*+45 deg*

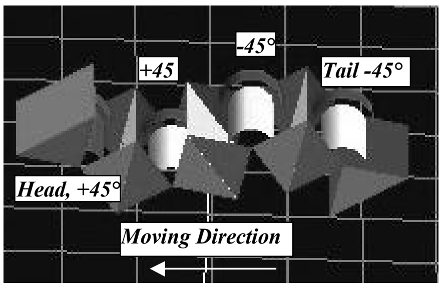


Fig. 4. Caterpillar tuple, propagation rule. Figure 10. Caterpillar gait, in a chain-typed modular robot, composed of four actuators

C = (state, angle)
P = (propagate hop-by-hop, storing on intermediate nodes changing the content accordingly to the table in figure 4-left. If on the head node and upon the receipt of a gait-tuple, re-apply propagation)

Each agent, upon the receipt of the caterpillar tuple, will drive the motor of its actuator to the angle in the content of the tuple. The coordination induced by the tuple leads the robot to the caterpillar gait as described in figure 4-right.

3.3 Ant-Based Routing on Mobile ad Hoc Networks

Routing protocols in wireless ad-hoc networks, inspired to the way in which ants collect food, have recently attracted the attention of the research community [7, 23]. Following this inspiration, the routing protocols build a sort of routing overlay structure (similar to ants' pheromone trials) by flooding the network and then exploit this overlaid structure for a much finer routing. We will show in this section how the basic mechanism of creating a routing overlay structure and the associated routing mechanism (similar to the ones already proposed in the area) can be effectively done within the TOTA model. The basic idea of the routing algorithm we will try to implement is the following [23]: when a node X wants to send a message to a node Y it injects a tuple representing the message to be sent, and a tuple used to create an overlay routing structure, for further use.

The tuple used to create the overlay structure can be described as follows:

C=("structure", nodeName, hopCount)
P=(propagate to all the nodes, increasing hopCount by one at every hop)

The tuple used to convey the message will be:

C=("message", sender,receiver,message)
P=(if a structure tuple having my same receiver can be found follow downhill its hopCount, otherwise propagate to all the nodes)

This routing algorithm is very simple: *structure* tuples create an overlay structure so that a *message* tuple following downhill a *structure* tuple's *hopCount* can reach the node that created that particular structure. In all situations in which such information is absent, the routing simply reduces to flooding the network. Although its simplicity, this model captures the basic underling model of several different MANET routing protocols [8]. The basic mechanism described in this section (tuples defining a structure to be exploited by other tuples' propagation) is fundamental in the TOTA approach and provides a great flexibility. For example it allows TOTA to realize systems such as CAN [24] and Pastry [25], to provide content-based routing in the Internet peer-to-peer scenario. In these models, peers forming an unstructured and dynamic community need to exchange data and messages not on the basis of the IP addressing scheme, but rather on the basis of the content of messages (e.g., "I need the mp3 of Hey Jude, no matter who can provide it to me"). To this end, these systems propose a communication mechanism based on a publish-subscribe model and rely on a properly built overlay space. A peer publishes information by sending them to a particular point of the overlaid space, while another read such information by looking for it in the same point of space (typically the process involves a hash function shared between all the peers, that maps keywords, associated to the information content, to points in space). TOTA can realize such systems by using a first layer of tuples defining the overlay space and then other tuples whose propagation rules let the tuples propagate efficiently in the overlaid space.

4 Performances and Experiments

One of the biggest concerns regarding our model is about scalability and performances. How much burden is requested to the system to maintain tuples?

Due to page limit, we will concentrate in this section to the kind of tuples used in the application examples. These are the tuples whose content depends only on the hop-distance from the source (*HopTuples*, from now on). Further details on these topics can be found in [17].

HopTuples' maintenance operations are required upon a change in the network topology, to have the distributed tuples reflect the new network structure. This means that maintenance operations are possibly triggered whenever, due to nodes' mobility or failures, new links in the network are created of removed. Because of scalability issues, it is fundamental that the tuples' maintenance operations are confined to an area neighboring the place in which the network topology had actually changed. This means that, if for example, a device in a MANET breaks down (causing a change in the network topology) only neighboring devices should change their tuples' values. The size of this neighborhood is not fixed and cannot be predicted a-priori, since it depends on the network topology.

For example, if the source of a tuple gets disconnected from the rest of the network, the updates must inevitably involve all the other peers in the network (that must erase that tuple form their repositories, see figure 5-top). However, especially for dense networks, this is unlikely to happen, and usually there will be alternative paths keeping up the tuple shape (see figure 5-bottom).

How can we perform such localized maintenance operations in a fully distributed way? To fix ideas, let us consider the case of a tuple incrementing its integer content by one, at every hop, as it is propagated far away from its source.

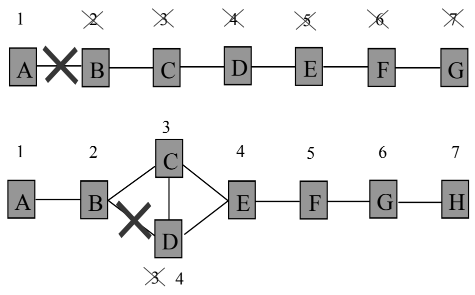


Fig. 5. The size of the update neighborhood depends on the network topology. Here is an example with a tuple incrementing its integer content by one, at every hop, as it is propagated far away from its source (top) the specific topology force update operations on the whole network (bottom) if alternative paths can be found, updates can be much more localized

Given a local instance of such a tuple X, we will call *Y a tuple that supports X* if: Y belongs to the same distributed tuple as X, Y is one-hop distant from X, Y value is equal to X value minus 1.With such a definition, a X's supporting tuple is a tuple that could have created X during its propagation.

Moreover, we will say that X is in a safe-state if it has a supporting tuple, or if it is the source of the distributed tuple. We will say that a tuple is not in a safe-state if the above condition does not apply.

Each local tuple can subscribe to the income or the removal of other tuples belonging to its same type in its one-hop virtual tuple space. This means, for example, that the tuple depicted in figure 5-bottom, installed on node F and having value 5 will be subscribed to the removal of tuples in its neighborhood (i.e. nodes E and G).

Upon a removal, each tuple reacts by checking if it is still in a safe-state. In the case a tuple is in a safe-state, the tuple the removal has not any effect - see later -. In the case a tuple is not in a safe state, it erases itself from the local tuple space. This eventually cause a cascading tuples' deletion until a safe-state tuple can be found, or the source is eventually reached, or all the tuples in that connected sub-network are deleted (as in the case of figure 5-top). When a safe-state tuple observe a deletion in its neighborhood it can fill that gap, and reacts by propagating to that node. This is what happens in figure 5-bottom, safe-state tuple installed on mode C and having value 3 propagates a tuple with value 4 to the hole left by tuple deletion (node D). It is worth noting that this mechanism is the same enforced when a new peer is connected to the network.

Similar considerations applies with regard to tuples' arrival: when a tuple sense the arrival of a tuple having value lower than its supporting tuple, it means that, because of nodes' mobility, a short-cut leading quicker to the source happened. Also in this case the tuple must update its value to take into account the new network topology.

So, what is the impact of a local change in the network topology in real scenarios?

To answer these questions we exploited the implemented TOTA emulator, being able to derive results depicted in figure 6.

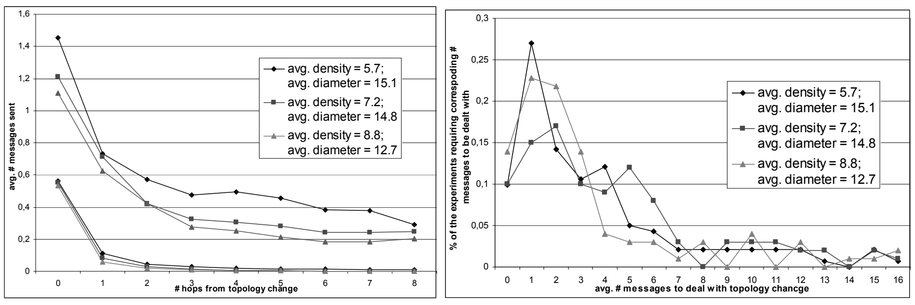


Fig. 6. Experimental results: locality scopes in tuple's maintenance operations emerge in a network without predefined boundaries. (left) topology changes are caused by random peer movements, on top, while topology changes are caused by the movement of the source peer on the bottom. (right) number of topology changes, happened during the experiments, that required a specific number of messages to be dealt with (e.g. in 27% of the diamond experiment, the topology change has been fixed with about 1.5 messages being exchanged)

The graphs show results obtained by more that 100 experiments, conducted on different networks. We considered networks having an average density (i.e. average number of nodes directly connected to an other node) of 5.7, 7.2 and 8.8 respectively (these numbers come from the fact that in our experiments they correspond to 150, 200, 250 peers, respectively). In each network, a tuple, incrementing its content at every hop, had been propagated. Nodes in the network move randomly, continuously changing the network topology. The number of messages sent between peers to keep the tuple shape coherent had been recorded.

Figure 6-left shows the average number of messages sent by peers located in an x-hop radius from the origin of the topology change. Figure 6-right shows the percentage of topology changes, happened during the experiments, that required a specific number of messages to be dealt with (see caption).

The most important consideration we can make looking at those graphs, is that, upon a topology change, a lot of update operations will be required near the source of the topology change, while only few operations will be required far away from it. This implies that, even if the TOTA network and the tuples being propagated have no artificial boundaries, the operations to keep their shape consistent are strictly confined within a locality scope (figure 6-left). Moreover, figure 6-right, show that the topology change that are likely to involve large-scale update are much less frequent than operations requiring only local rearrangements. This fact supports the feasibility of the TOTA approach in terms of its scalability. In fact, this means that, even in a large network with a lot of nodes and tuples, we do not have to continuously flood the whole network with updates, eventually generated by changes in distant areas of the network. Updates are almost always confined within a locality scope from where they took place.

5 Related Work

Several proposals in the last years are challenging the traditional ideas and methodologies of software engineering and inspired to physical, biological models are entering in the distributed application and multi agent system research frameworks.

An area in which the problem of achieving context-awareness and adaptive coordination has been effectively addressed (and that, consequently, has partially influenced our proposal) is amorphous and paintable computing [9, 19]. The particles constituting an amorphous computer have the basic capabilities of propagating sorts of computational fields in the network, and to sense and react to such fields. In particular, particles can transfer an activity state towards directions described by fields' gradients, so as to make coordinated patterns of activities (to be used for, e.g. self-assembly) emerge in the system independently of the specific structure of the network (which is, by definition, amorphous). Similarly with TOTA, such an approach enables, via the single abstraction of fields, to both diffuse contextual information and to organize adaptive global coordination patterns. The main difference between TOTA and this approach is the application domain: TOTA is not only addressed to amorphous networks of nano- or micro-devices, but it aims also to address networks of mobile devices like cellular phones, PDA and laptops. Moreover, because of this difference, one of the TOTA main concerns, that is totally neglected in amorphous computer, is the need to constantly manage distributed tuples' values so as to maintain their intended shape despite network reconfigurations.

Anthill [2] is a framework built to support design and development of adaptive peer-to-peer applications, that exploits an analogy with biological adaptive systems [7, 20]. Anthill consists of a dynamic network of peer nodes, each one provided with a local tuple space ("nest"), in which distributed mobile components ("ants") can travel and can indirectly interact and cooperate with each other by leaving and retrieving tuples in the distributed tuple spaces. The key objective of anthill is to build robust and adaptive networks of peer-to-peer services (e.g., file sharing) by exploiting the capabilities of ants to re-shape their activity patterns accordingly to the changes in the network structure. Although we definitely find the idea interesting and promising, a more general flexible approach would be needed to support – other than adaptive resource sharing – adaptive coordination in distributed applications.

The popular videogame "The Sims" [28] exploits sorts of computational fields, called "happiness landscapes" and spread in the virtual city in which characters live, to drive the movements of non-player characters. In particular, non-player characters autonomously move in the virtual Sims city with the goal of increasing their happiness by climbing the gradients of specific computational fields. For instance, if a character is hungry, it perceives and follows a happiness landscape whose peaks correspond to places where food can be found, i.e., a fridge. After having eaten, a new landscape will be followed by the character depending on its needs. Although sharing the same inspiration, "Sims' happiness fields" are static and generated only by the environment. In TOTA, instead, tuples are dynamic and can change over time, and agents themselves are able to inject tuples to promote a stronger self-organization perspective.

The MMASS formal model for multi-agent coordination, described in [3], represents the environment as a multi-layered graph in which agents can spread abstract fields representing different kinds of stimuli through the nodes of this graph. The agents' behavior is then influenced by the stimuli they perceive in their location. In fact agents can associate reactions to these stimuli, like in an event-based model, with the add-on of the location-dependency that is associated to events and reactions. The main difference between MMASS and TOTA the application domain: MMASS is mainly devoted to simulation of artificial societies and social phenomena, thus its main implementation is based on cellular automata, TOTA is mainly interested in distributed (pervasive) computing and, accordingly, its implementation is based on real devices forming wireless networks.

The L2imbo model, proposed in [11], is based on the notion of distributed tuple spaces augmented with processes (Bridging Agents) in charge of moving tuples form one space to another. Bridging agent can also change the content of the tuple being moved for example to provide format conversion between tuple spaces. The main differences between L2imbo and TOTA are that in L2imbo, tuples are conceived as "separate" entities and their propagation is mainly performed to let them being accessible from multiple tuple spaces. In TOTA, tuples form distributed data structure and their "meaning" is in the whole data structure rather than in a single tuple. Because of this conceptual difference, tuples' propagation is defined for every single tuple in TOTA, while is defined for the whole tuple space in L2imbo.

6 Conclusions and Future Work

Tuples On The Air (TOTA) promotes programming distributed applications by relying on distributed data structures, spread over a network as sorts of electromagnetic fields, and to be used by application agents both to extract contextual information and to coordinate with each other in an effective way. As we have tried to show in this paper, TOTA tuples support coordination and self-organization, by providing a mechanism to both enable agents interactions and to represent contextual information in a very effective way.

Despite the fact there are a lot of examples we had been able to realize with TOTA, we still do not have a general engineering methodology or primitive tuples' types on which to build and generalize other kind of applications. However this is not our specific limit, but it is a current general limitation: a general methodology for dealing with bottom up approaches (like the one promoted by TOTA) is still unknown. However, we think that such methodology could be found in the future and for sure, our final goal would be to develop a complete engineering procedure for this kind of model. In pursuing this goal, deployment of applications will definitely help identifying current shortcomings and directions of improvement. In particular our future work will be based on applying the TOTA model, in the development of new applications for sensor networks with a particular interest in those algorithms exploiting ideas taken from manifold geometry [30]. From a more pragmatic perspective, much more performance evaluations are needed to test the limits of usability and the scalability of TOTA by quantifying the TOTA delays in updating the tuples' distributed structures in response to dynamic changes.

Acknowledgements

This work is supported by the Italian MIUR and CNR in the "Progetto Strategico IS-MANET, Infrastructures for Mobile ad-hoc Networks ".

References

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman and R. Weiss, "Amorphous Computing", Communications of the ACM, 43(5), May 2000.

[2] O. Babaoglu, H. Meling, A. Montresor, "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems", in Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS '02), Vienna, Austria, July 2002.

[3] S. Bandini, S. Manzoni, C. Simone, "Space Abstractions for Situated Multiagent Systems", 1st International Joint Conference on Autonomous Agents and Multiagent Systems, Bologna (I), ACM Press, pp. 1183-1190, July 2002.

[4] Y. Bar-Yam. Dynamics of Complex systems. Addison-Wesley, 1997.

[5] L. Barabasi, "Linked", Perseus Press, 2002.

[6] F. Bellifemine, A. Poggi, G. Rimassa, "JADE - A FIPA2000 Compliant Agent Development Environment", 5th International Conference on Autonomous Agents (Agents 2001), pp. 216-217, Montreal, Canada, May 2001.

[7] E. Bonabeau, M. Dorigo, G. Theraulaz, "Swarm Intelligence", Oxford University Press, 1999.

[8] J. Broch, D. Maltz, D. Johnson, Y. Hu, J. Jetcheva, "A Perfomance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", ACM/IEEE Conference on Mobile Computing and Networking, Dallas (TX), Oct. 1998.

[9] W. Butera, "Programming a Paintable Computer", PhD Thesis, MIT Media Lab, Feb. 2002.

[10] G. Cabri, L. Leonardi, F. Zambonelli, "Engineering Mobile Agent Applications via Context-Dependent Coordination", IEEE Transactions on Software Engineering, 28(11):1040:1058, Nov. 2002.

[11] N. Davies, A. Friday, S. Wade, G. Blair, "L2imbo: A distributed systems platform for mobile computing", ACM Mobile Networks and Applications, 3(2):143-156, Aug., 1998.

[12] D. Gelernter, N.Carriero "Coordination Languages and Their Significance", Communication of the ACM, 35(2):96-107, Feb. 1992.

[13] J. Kephart, D. M. Chess, "The Vision of Autonomic Computing", IEEE Computer, 36(1):41-50, Jan. 2003.

[14] D. Loomis, "The TINI Specification and Developer's Guide", http://www.ibutton.com/TINI/book.html.

[15] M. Mamei, L. Leonardi, M. Mahan, F. Zambonelli, "Coordinating Mobility in a Ubiquitous Computing Scenario with Co-Fields", Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices, AAMAS 2002, Bologna, Italy, July 2002.

[16] M. Mamei, Franco Zambonelli, Letizia Leonardi, "Tuples On The Air: a Middleware for Context-Aware Computing in Dynamic Networks", 1st International Workshop on Mobile Computing Middleware at the 23rd International Conference on Distributed Computing Systems (ICDCS), IEEE CS Press, pp. 342-347, Providence (RI), USA, May 2003.

[17] M. Mamei, F. Zambonelli, "Self-Maintained Distributed Data Structure Over Mobile Ad-Hoc Network", Technical Report No. DISMI-2003-23, University of Modena and Reggio Emilia, August 2003.
[18] S. Mostefaoui, O. Rana, N. Foukia, S. Hassas, G. Serugendo, C. Van Aart, A. Karageorgos, “Self-Organizing Applications: a Survey”, in this volume.
[19] R. Nagpal, A. Kondacs, C. Chang, “Programming Methodology for Biologically-Inspired Self-Assembling Systems”, in the AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality, March 2003.
[20] V. Parunak, S. Bruekner, J. Sauter, "ERIM's Approach to Fine-Grained Agents", NASA/JPL Workshop on Radical Agent Concepts, Greenbelt (MD), Jan. 2002.
[21] G. P. Picco, A. L. Murphy, G. C. Roman, “LIME: a Middleware for Logical and Physical Mobility”, In Proceedings of the 21st International Conference on Distributed Computing Systems, IEEE CS Press, July 2001.
[22] Polybot Simulator, downloadable from: http://www2.parc.com/spl/projects/modrobots/simulations/index.html
[23] R. Poor, “Embedded Networks: Pervasive, Low-Power, Wireless Connectivity", PhD Thesis, MIT, 2001.
[24] S. Ratsanamy,, P. Francis, M. Handley, R. Karp, "A Scalable Content-Addressable Network", ACM SIGCOMM Conference 2001, San Diego (CA), ACM Press, Aug. 2001.
[25] A. Rowstron, P. Druschel, “Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems”, 18th IFIP/ACM Conference on Distributed Systems Platforms, Heidelberg (D), Nov. 2001.
[26] D. Servat, A. Drogoul, “Combining amorphous computing and reactive agent-based systems: a paradigm for pervasive intelligence?”, AAMAS, Bologna (I), July, 2002.
[27] W. Shen, B. Salemi, P. Will, “Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots”, IEEE Transactions on Robotics and Automation 18(5):1-12, Oct. 2002.
[28] The Sims, http://thesims.ea.com
[29] M. Yim, Y. Zhang, D. Duff, “Modular Robots”, IEEE Spectrum Magazine, February 2002.
[30] F. Zambonelli, M. Mamei, “The Cloak of Invisibility: Challenges and Applications”, IEEE Pervasive Computing, 1(4):62-70, Oct.-Dec. 2002.
[31] F. Zambonelli, V. Parunak, “From Design to Intention: Signs of a Revolution”, 1st Intl. ACM Conference on Autonomous Agents and Multiagent Systems”, Bologna (I), July 2002.

Providing Effective Access to Shared Resources: A COIN Approach

Stéphane Airiau[1], Sandip Sen[1], David H. Wolpert[2], and Kagan Tumer[2]

[1] University of Tulsa
Mathematical and Computer Sciences Department
600 South College avenue, Tulsa, OK 74104
{stephane,sandip-sen}@utulsa.edu
[2] NASA Ames Research Center
M/S 269-2, Moffett Field, CA 94035
{dhw,kagan}@email.arc.nasa.gov

Abstract. Managers of systems of shared resources typically have many separate goals. Examples are efficient utilization of the resources among its users and ensuring no user's satisfaction in the system falls below a preset minimal level. Since such goals will usually conflict with one another, either implicitly or explicitly the manager must determine the relative importance of the goals, encapsulating that into an overall utility function rating the possible behaviors of the entire system. Here we demonstrate a distributed, robust, and adaptive way to optimize that overall function. Our approach is to interpose adaptive agents between each user and the system, where each such agent is working to maximize its own private utility function. In turn, each such agent's function should be both relatively easy for the agent to learn to optimize, and "aligned" with the overall utility function of the system manager — an overall function that is based on but in general different from the satisfaction functions of the individual users. To ensure this we enhance the COllective INtelligence (COIN) framework to incorporate user satisfaction functions in the overall utility function of the system manager and accordingly in the associated private utility functions assigned to the users' agents. We present experimental evaluations of different COIN-based private utility functions and demonstrate that those COIN-based functions outperform some natural alternatives.

1 Introduction

One of the key problems confronting the designers of large-scale, distributed agent applications is control of access to shared resources. In order for the system to function well, access to these shared resources must be coordinated to eliminate potential problems like deadlock, starvation, livelock and other forms of resource contention. Without proper coordination, both individual user satisfaction and overall system performance can be significantly impaired.

As hand-coded, static procedures are likely to be brittle in complex environments, we are interested in flexible, adaptive, scalable approaches to the resource

© Springer-Verlag Berlin Heidelberg 2004

sharing problem [1]. One approach is to use distributed machine learning-based agents each of which works exclusively to increase the satisfaction of its associated user. While this is appealing, individual learners optimizing the "satisfaction" utility functions of their users typically will not coordinate their activities and may even work at cross-purposes, thereby degrading the performance of the entire system. Well-known cases of this problem of global shared resources include the Tragedy of the Commons [2]. Such problems highlight the need for careful design of the utility functions each of the learning agents work to optimize. In general, this means having each agent use a utility function that differs from the satisfaction function of its associated user [3]. In addition to ensuring that the private utility functions of the agents do not induce them to work at cross-purposes though, it is also necessary that those functions can be readily optimized in an adaptive manner, despite high noise levels in the environment.

The field of mechanism design, originating in game theory and economics, appears to address this problem. However it suffers from restrictions that diminish its suitability for problems — like the ones considered here — involving bounded-rational, non-human, noisy agents, where the variables controlled by the agents and even the number of agents can be varied [4]. In contrast, the COllective INtelligence (COIN) framework is explicitly formulated to address such problems without such restrictions [3, 5, 6]. In particular, its mathematics derives private utility functions to provide the individual learning agents that are *learnable*, in addition to inducing the agents not to work at cross-purposes. Such functions are designed both so that the agents can perform well at maximizing them, and so that as they do this the agents also "unintentionally" improve the provided "world utility" function rating behavior of the entire system.

In the past, COIN techniques have been used for world utility functions that are fully exogenous, in that they do not reflect any satisfaction functions of a set of users of the system nor how best to accommodate those satisfaction functions. While achieving far superior performance in the domains tested to date than do conventional approaches like greedy agents and team games [7, 5, 8], these tests do not assess how well COIN-based systems perform in situations like shared resource problems, in which the world utility is not fully exogenous. In this paper we start by reviewing the mathematics of collective intelligence. We then test the techniques recommended by that mathematics on shared resource problems, using a world utility function that reflects the satisfaction levels of the individual users of the system, and in particular trades off concern that no individual's satisfaction be too low with concern that the overall system behave efficiently. We demonstrate that in such domains COIN techniques also far outperform approaches like team games. We also validate some quantitative predictions of the COIN mathematics concerning how the world utility is affected by various modifications to the COIN techniques.

2 Background on Collective Intelligence

The "COllective INtelligence" framework (COIN) concerns the design of large distributed collectives of self-interested agents where there exists a world utility function measuring the performance of the entire collective. The emphasis of the research is on the inverse problem: given a set of self-interested agents and a world utility function, how should a designer configure the system, and in particular set the private utility functions of the agents so that, when all agents try to optimize their functions, the entire collective performs better. In this section, we introduce the mathematics of designing collectives. Because of space limitations, the concepts cannot be justified or elaborated in any detail; references to other papers with such details are provided.

2.1 Central Equation

Let ζ be an arbitrary space whose elements z give the joint move of all agents in the collective system. We wish to search for the z that maximizes the provided **world utility** function $\mathcal{G}(z)$. In addition to $\mathcal{G}$, for each agent η controlling z_η, there is a private utility function $\{g_\eta\}$. We use the notation $\hat{}\,\eta$ to refer to all agents other than η. The agents act to improve their private utility functions, even though we are only concerned with the value of the world utility $\mathcal{G}$.

We need a way to "calibrate" the utility functions so that the value assigned to a joint action z reflects the ranking of z relative to all the other possible joint actions in ζ. We denote as intelligence the result of this calibration process. Intuitively, the intelligence indicate what percentage of η's actions would have resulted in lower utility (if 99% of the available actions lead to a worse utility, the agent made a smart decision). In the COIN framework, the "intelligence for η at z with respect to U" is defined by:

$$N_{\eta,U}(z) \equiv \int d\mu_{z_{\hat{}\eta}}(z')\Theta[U(z) - U(z')] \ , \tag{1}$$

where Θ is the Heaviside function [1], and where the subscript on the (normalized) measure $d\mu$ indicates it is restricted to z' sharing the same non-η components as z. There is no particular constraint on the measure μ other than reflecting the type of system (whether ζ is countable or not, if not, what coordinate system is being used...).

As system designer, our uncertainty concerning the state of the system is reflected in a probability distribution P over ζ. Our ability to control the system consists of setting the value of some characteristic of the collective, which is denoted as its **design coordinate**. For instance, the design coordinate can be setting the private utility functions of the agents. For a value s of the design coordinate, our analysis revolves around the following **central equation** for $P(\mathcal{G} \mid s)$, which follows from Bayes' theorem:

[1] The Heaviside function is defined to equal 1 when its argument is greater or equal to 0, and to 0 otherwise.

$$P(\mathcal{G} \mid s) = \underbrace{\int d\boldsymbol{N}_{\mathcal{G}} P(\mathcal{G} \mid \boldsymbol{N}_{\mathcal{G}}, s)}_{\text{explore vs. exploit}} \int \underbrace{d\boldsymbol{N}_{g} P(\boldsymbol{N}_{\mathcal{G}} \mid \boldsymbol{N}_{g}, s)}_{\text{factoredness}} \underbrace{P(\boldsymbol{N}_{g} \mid s)}_{\text{learnability}} \ , \qquad (2)$$

where $\boldsymbol{N}_g$ and $\boldsymbol{N}_{\mathcal{G}}$ are the **intelligence** vectors of the agents with respect to the private utility g_η of η and the world utility $\mathcal{G}$, respectively.

Note that $N_{\eta,g_\eta}(z) = 1$ means that agent η is fully rational at z, in that its move maximizes the value of its utility, given the moves of the agents. In other words, a point z where $N_{\eta,g_\eta}(z) = 1$ for all agents η is one that meets the definition of a game-theory Nash equilibrium. On the other hand, a z at which all components of $\boldsymbol{N}_{\mathcal{G}} = 1$ is a maximum $\mathcal{G}$ along all coordinates of z. So if we can get these two points to be identical, then if the agents do well enough at maximizing their private utilities we are assured we will be near an axis-maximizing point for $\mathcal{G}$.

To formalize this, consider our decomposition of $P(\mathcal{G} \mid s)$.

- **learnability:** if we can choose the global coordinate s so that the third conditional probability in the integrand is peaked around vectors $\boldsymbol{N}_g$ all of whose components are close to 1 (that is agents are able to learn their tasks), then we have likely induced large (private utility function) intelligences. Intuitively, this ensures that the private utility functions have high "signal-to-noise".
- **factoredness**: by choosing s, if we can also have the second term be peaked about $\boldsymbol{N}_{\mathcal{G}}$ equal to $\boldsymbol{N}_g$ (that is the private utility and the world utility are aligned), then $\boldsymbol{N}_{\mathcal{G}}$ will also be large. It is in the second term that the requirement that the private utility functions be "aligned with $\mathcal{G}$" arises. Note that our desired form for the second term in Equation 2 is assured if we have chosen private utilities such that $\boldsymbol{N}_g$ equals $\boldsymbol{N}_{\mathcal{G}}$ exactly for all z. Such a system is said to be **factored**.
- Finally, if the first term in the integrand is peaked about high $\mathcal{G}$ when $\boldsymbol{N}_{\mathcal{G}}$ is large, then our choice of s will likely result in high $\mathcal{G}$, as desired.

On the other hand, unless one is careful, each agent will have a hard time discerning the effect of its behavior on its private utility when the system is large. Typically, each agent has a horrible "signal-to-noise" problem in such a situation. This will result in a poor term three in the central equation. For instance, consider a collective provided by the human economy and a "team game" in which all the private utility functions equal $\mathcal{G}$. Every citizen gets the national GDP as her/his reward signal, and tries to discern how best to act to maximize that reward signal. At the risk of understatement, this would provide the individual members of the economy with a difficult reinforcement learning task.

In this paper, we concentrate on the second and third terms, and show how to simultaneously set them to have the desired form in the next section. Hence, the research problem is to choose s so that the system both has good learnabilities for its agents, and is factored.

Mechanism design might, at first glance, appear to provide us techniques for solving this version of the inverse problem. However while it can be viewed as addressing the second term, the issue of learnability is not studied in mechanism design. Rather mechanism design is almost exclusively concerned with collectives that are at (a suitable refinement of) an exact Nash equilibrium [9]. That means that every agent is assumed to be performing *as well as is theoretically possible*, given the behavior of the rest of the system. In setting private utilities and the like on this basis, mechanism design ignores completely the issue of how to design the system so that each of the agents can achieve a good value of its private utility (given the behavior of the rest of the system). In particular it ignores all statistical issues related to how well the agents can be expected to perform for various candidate private utilities.

Such issues become crucial as one moves to large systems, where each agent is implicitly confronted with a very high-dimensional reinforcement learning task. It is its ignoring of this issue that means that mechanism design scales poorly to large problems. In contrast, the COIN approach is precisely designed to address both learnability issues as well as term 2.

2.2 Wonderful Life Utility and Aristocrat Utility

As an example of the foregoing, any "team game" in which all private utility functions equal $\mathcal{G}$ is factored [10]. However as illustrated above in the human economy example, team games often have very poor forms for term 3 in Equation 2, forms which get progressively worse as the size of the collective grows. This is because for such private utility functions each agent η will usually confront a very poor "signal-to-noise" ratio in trying to discern how its actions affect its utility $g_\eta = \mathcal{G}$, since so many other agent's actions also affect $\mathcal{G}$ and therefore dilute η's effect on its own private utility function.

We now focus on algorithms based on private utility functions $\{g_\eta\}$ that optimize the signal/noise ratio reflected in the third term of the central equation, subject to the requirement that the system be factored. We will introduce two utility functions satisficing this criteria, namely the Wonderful Life Utility (WLU) and the Aristocrat Utility (AU).

To understand how these algorithms work, say we are given an arbitrary function $f(z_\eta)$ over agent η's moves, two such moves ${z_\eta}^1$ and ${z_\eta}^2$, a utility U, a value s of the design coordinate, and a move by all agents other than η, $z_{\hat{}\eta}$. Define the associated **learnability** by

$$\Lambda_f(U; z_{\hat{}\eta}, s, {z_\eta}^1, {z_\eta}^2) \equiv \sqrt{\frac{[E(U; z_{\hat{}\eta}, {z_\eta}^1) - E(U; z_{\hat{}\eta}, {z_\eta}^2)]^2}{\int dz_\eta [f(z_\eta) Var(U; z_{\hat{}\eta}, z_\eta)]}} \; . \tag{3}$$

The expectation values in the numerator are formed by averaging over the training set of the learning algorithm used by agent η, n_η. Those two averages are evaluated according to the two distributions $P(U|n_\eta)P(n_\eta|z_{\hat{}\eta}, {z_\eta}^1)$

and $P(U|n_\eta)P(n_\eta|z_{\hat{}\eta}, {z_\eta}^2)$, respectively. (That is the meaning of the semicolon notation.) Similarly the variance being averaged in the denominator is over n_η according to the distribution $P(U|n_\eta)P(n_\eta|z_{\hat{}\eta}, z_\eta)$.

The denominator in Equation 3 reflects how sensitive $U(z)$ is to changing $z_{\hat{}\eta}$. In contrast, the numerator reflects how sensitive $U(z)$ is to changing z_η. So the greater the learnability of a private utility function g_η, the more $g_\eta(z)$ depends only on the move of agent η, i.e., the better the associated signal-to-noise ratio for η. Intuitively then, so long as it does not come at the expense of decreasing the signal, increasing the signal-to-noise ratio specified in the learnability will make it easier for η to achieve a large value of its intelligence. This can be established formally: if appropriately scaled, g'_η will result in better expected intelligence for agent η than will g_η whenever $\Lambda_f(g'_\eta; z_{\hat{}\eta}, s, {z_\eta}^1, {z_\eta}^2) > \Lambda_f(g_\eta; z_{\hat{}\eta}, s, {z_\eta}^1, {z_\eta}^2)$ for all pairs of moves ${z_\eta}^1, {z_\eta}^2$[6].

One can solve for the set of all private utilities that are factored with respect to a particular world utility. Unfortunately though, in general a collective cannot both be factored and have infinite learnability for all of its agents [6]. However consider **difference** utilities, of the form

$$U(z) = \beta[\mathcal{G}(z) - D(z_{\hat{}\eta})] \tag{4}$$

Any difference utility is factored [6]. In addition, for all pairs ${z_\eta}^1, {z_\eta}^2$, under benign approximations, the difference utility maximizing $\Lambda_f(U; z_{\hat{}\eta}, s, {z_\eta}^1, {z_\eta}^2)$ is found by choosing

$$D(z_{\hat{}\eta}) = E_f(\mathcal{G}(z) \mid z_{\hat{}\eta}, s) \;, \tag{5}$$

up to an overall additive constant, where the expectation value is over z_η. We call the resultant difference utility the **Aristocrat** utility (AU), loosely reflecting the fact that it measures the difference between a agent's actual action and the average action. If each agent η uses an appropriately rescaled version of the associated AU as its private utility function, then we have ensured good form for both terms 2 and 3 in Equation 2.

Using AU in practice is sometimes difficult, due to the need to evaluate the expectation value. Fortunately there are other utility functions that, while being easier to evaluate than AU, still are both factored and possess superior learnability to the team game utility, $g_\eta = \mathcal{G}$. One such private utility function is the **Wonderful Life** Utility (WLU). The WLU for agent η is parameterized by a pre-fixed **clamping parameter** CL_η chosen from among η's possible moves:

$$WLU_\eta \equiv \mathcal{G}(z) - \mathcal{G}(z_{\hat{}\eta}, CL_\eta) \;. \tag{6}$$

WLU is factored no matter what the choice of clamping parameter. Furthermore, while not matching the high learnability of AU, WLU usually has far better learnability than does a team game, and therefore (when appropriately scaled) results in better expected intelligence [3, 5, 6].

3 Problem Definition

Let consider a set of users and a set of m shared resources/machines. Different users have different task loads, each task being of one of T types. Any machine can perform any task type, but different machines have different processing speeds for each task type. A given user sends all of its tasks of a particular type to a specific machine but can send tasks of different types to different machines. So each user must decide, for each $j \in T$, to what machine A_i^j to send all of its tasks of type j.

We consider a batch mode scenario, in which every user submits all of its tasks, the machines complete their tasks, and the associated overall performance of the system is ascertained. Each machine works by grouping all the tasks sent to it by type, and performs all tasks of one type contiguously before returning them to the associated users and then switching to the next task type. The order of processing different task types is fixed for any given machine. Each type is performed at a speed specific to the machine.

3.1 Satisfaction Functions of Users

User i has a personal "satisfaction" utility function, $\mathcal{H}_i$, which is an inverse function of the time that the user has to wait for all of his tasks to finish, i.e is an inverse function of the delay. The user may be less willing to wait for certain task types compared to others. The user can provide feedback or her/his model of satisfaction to the system manager. We indicate the completion time of the tasks of type j submitted by user i to machine A_i^j by CT_i^j.

The functions defined below measure dissatisfaction rather than satisfaction, in that they are monotonically increasing functions of the delay. So the goal for agent i is to maximize $\mathcal{H}_i = -\mathcal{D}_i$, where $\mathcal{D}_i$ is defined as one of the following:

- the maximum delay for user i, $\mathcal{D}_i = \max_{j \in \{1..T\}} CT_i^j$,
- an importance-weighted combination of the time of completion of all the tasks of the user, $\mathcal{D}_i = \sum_{j=1}^{T} \alpha_j CT_i^j$.

$\mathcal{D} \equiv \{\mathcal{D}_i\}$ is the set of the dissatisfactions of all users.

3.2 Definition of the World Utility Function $\mathcal{G}$

The world utility function measures the performance of the overall system. The system designer chooses this function and might evaluate performance by measuring some characteristics of resource usage (e.g., the load distribution or the idle time). The system designer might also incorporate user preferences to evaluate the performance of the system. We have focused on this aspect by defining the world utility function as a function of $\mathcal{D}$, the dissatisfactions of all users. This assumes that either the users have provided a model of their preferences to the system designer or that the system designer has modeled them. In the former case, users may be allowed to update their model.

We have experimented with the following world utilities:

- Avoid hurting any user: minimize $\mathcal{G}(\mathcal{D}_i) = \max_i \mathcal{D}_i$.
- Minimize a linear combination of the satisfaction of individual users where the weight associated with a user represents the importance of that particular user: $\mathcal{G}(\mathcal{D}_i) = \sum_i w_i \mathcal{D}_i$.
- To have high average satisfaction without any user's satisfaction being low, minimize $\mathcal{G}(\mathcal{D}_i) = \beta * \sigma_{\mathcal{D}} + \sum_i w_i \mathcal{D}_i$, $\sigma_{\mathcal{D}}$ being the variance in the dissatisfactions.

3.3 Agent Learning Algorithms

Once the measure of dissatisfaction of each agent and the performance metric of the overall system have been defined, the remaining question is how to improve the performance. In the experiments, we want to answer the question: what is the function that each self interested agent has to improve? We refer to this function as the **private utility** for the agent. In other words, given the agents' satisfaction model and the performance metric of the system, we want to be able to tell the agents that the best way to improve the performance of the system (which include their own preferences) is to optimize a certain private utility. If they decide to optimize other criteria, the performance of the system will not be as good as if they follow the recommendation.

We compared performance ensuring from four private utility functions for the agents, the last two being those recommended by the COIN framework:

- a **team game**; each agent's utility equals $\mathcal{G}$, i.e. by choosing its action, each agent is trying to optimize the performance of the overall system (the agent only focuses on the result of the team, it does not consider its own performance);
- a **greedy game**: each agent i's private utility is $\mathcal{H}_i$ (the agent focuses only on its performance);
- **AU**: to compute AU for an agent $i \in U$, we need to evaluate $\mathcal{G} - \sum_{j=1}^{T} \mathcal{G}(i, j)$ where $\mathcal{G}(i, j)$ denotes the world utility when agent i sends its job to machine j while all other agents send their jobs to the same machines they actually used. Thus, we need to re-evaluate the completion times of all machines m when the other agents maintain their decision while agent i sending its load to m). See [6] for a discussion of efficient evaluation of AU;
- **WLU**: to compute WLU for agent i, we chose to clamp to "null" the action of i. Hence, $WLU_i = \mathcal{G} - \mathcal{G}(i, \emptyset)$. Thus we only need to evaluate what the completion time of the machine chosen by i would be if η did not send any load to that machine. See [6] for a discussion of efficient evaluation of AU.

We had each agent use an extremely simple reinforcement learning algorithm in our experiments, so that performance more accurately reflects the quality of the agents' utility functions rather than the sophistication of the learning scheme. In this paper, each agent is "blind", knowing nothing about the environment in which it operates, and observing only the utility value it gets after an iteration of the batch process. The agents each used a modified version a *Boltzmann learning* *algorithm* to map the set of such utility values to a probability distribution over the (finite) set of possible moves, a distribution that is then sampled to select the agent's move for the next iteration. Each agent tries to learn the estimated reward $\bar{R}_i$ corresponding to the action i. An agent will choose the action i with probability $\mathcal{P}_i = \frac{e^{-\beta \bar{R}_i}}{\sum_{\text{all actions j}} e^{-\beta \bar{R}_j}}$. The parameter β is the *inverse (Boltzmann) temperature*. During round r, if the action i is chosen, the update of the estimated reward for action i is $\bar{R}_i = \sum_{j \in \mathcal{C}_i} \frac{e^{-\alpha(r-j)}}{\sum_{k \in \mathcal{C}_i} e^{-\alpha(r-k)}} R_j$, where $\mathcal{C}_i$ denotes the set of rounds where action i was chosen, and R_j denotes the reward received at the end of the round j. The *data aging* parameter α models the importance of the choices made in the previous rounds and enables to speed up the learning process. In our modification, rather than have the expected utility value for each possible move of an agent be a uniform average of the utilities that arose in the past when it made that move, we exponentially discount moves made further into the past, to reflect the non-stationarity of the system.

4 Experiments

We experiment with a domain where users are sending different types of printing jobs to a shared pool of heterogeneous printers. The different types of printing jobs may represent different kind of printing operations (for instance, printing black & white, printing in color) Though each printer is able to process any job type, a given printer processes jobs of different types at different speeds. For each printer, we randomly choose the speed of processing each task type, and processing occurs in the order of decreasing speed.

Each user has a load of printing jobs to perform. We assign one agent to handle all tasks of a given type for a given user, and its job is to select a printer to which these tasks would be sent. Each run starts with an initial sequence of iterations in which the agents make purely random moves to generate data for the learning algorithms. We then successively "turn on" more and more of those algorithms i.e., at each successive iteration, a few more agents start to make their moves based on their learning algorithm and associated data rather than randomly. We start each agent with a high *Boltzmann temperature* $\frac{1}{\beta}$, usually 10, and decrease it at each iteration by multiplying it by a decay factor. We refer to a temperature schedule as *slow* when the decay factor is large (for instance 99%), and refer to the schedule as *fast* when the decay factor is small (for instance 70%). We use an aging parameter in forming expectation values of 0.1.

In the experiments reported here, the dissatisfaction of a given user is $\mathcal{D}_i = \sum_{j=1}^{T} \alpha_j CT_i^j$ where the α_j are randomly chosen and remain the same throughout the learning process. We used 10 users, 8 types and 6 machines.

4.1 Results

We first considered the case where the World Utility Function $\mathcal{G}$ is computed as the maximum of the dissatisfaction of the user, i.e., $\mathcal{G} = \max_i \mathcal{D}_i$. The goal

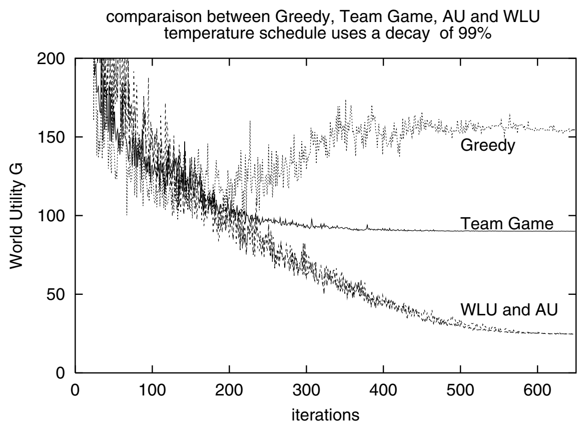


Fig. 1. Comparison between different agent utilities when $\mathcal{G} = \max_i \mathcal{D}_i$

is to minimize the maximum dissatisfaction among the users. The decay factor used is 0.99. In Fig. 1 we provide performance for the different agent utility functions averaged over 10 runs. The performances achieved by AU and WLU are similar, and dominate that of both Team Game and Greedy agent utilities. We found that at the end of the learning process, the standard deviation in $\mathcal{G}$ for AU and WLU is small (respectively 2.1 and 1.08) whereas it is large for Greedy and Team Game (respectively 14.4 and 24). This corroborates the reasoning that signal to noise is too large when we use Greedy or Team Game, and therefore the learning agents are not able to determine the impact of their actions on their utility functions. In contrast, as discussed above, AU and WLU are designed to have good signal to noise, which enables them to reach better (and more consistent) performance levels.

Note that with the greedy approach $\mathcal{G}$ *worsens* as each agent learns how to maximize its utility. This is an example of a tragedy of the commons: since the Greedy utility is not factored with respect to $\mathcal{G}$, the associated Nash equilibrium — achieved as the agents learn more — has poor $\mathcal{G}$.

In Fig. 2, $\mathcal{G}$ is instead the weighted sum of the dissatisfaction of the users, i.e., $\mathcal{G} = \sum_i w_i \mathcal{D}_i$. In figure 3 it is the sum of all the users dissatisfactions with the standard deviation of those dissatisfactions, i.e., $\mathcal{G} = \beta * \sigma_{\mathcal{D}} + \sum_i \mathcal{D}_i$ (we have used $\beta = 1$). The decay factor used in these experiments is 0.99. In all cases, AU and WLU achieve comparable performance, and outperform team game.

Although the asymptotic performance of the team game is not as good as that of AU or WLU, in both of these experiments team game performs better than AU or WLU in the early phase of the learning. In addition the convergence in time of $\mathcal{G}$ for AU and WLU is slower in these experiments. This is consistent with COIN theory, which says that simply changing to a more learnable utility function may actually lead to *worse* performance if the learning temperature is not changed simultaneously. Intuitively, to achieve their better signal to noise

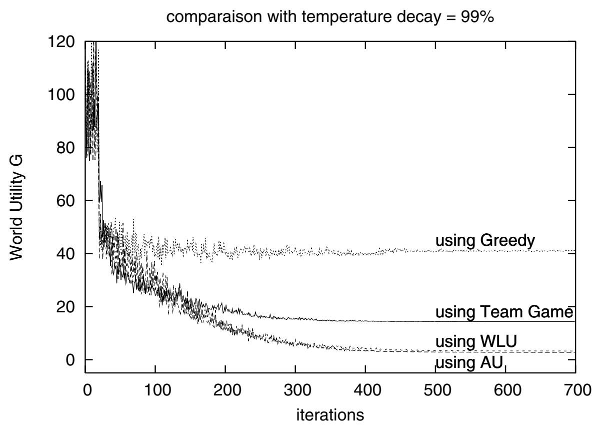


Fig. 2. Comparison between different reward in the case where $\mathcal{G} = \sum_i w_i \mathcal{D}_i$

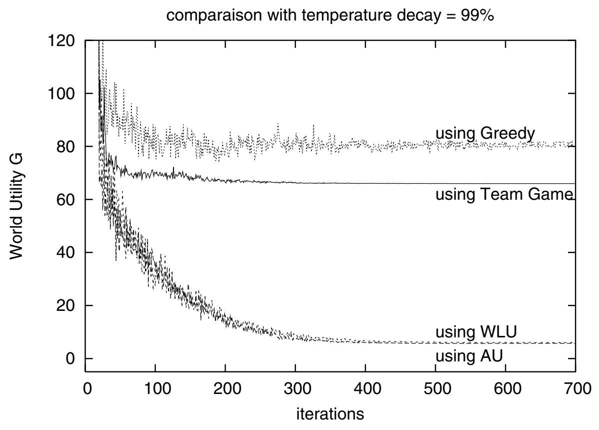


Fig. 3. Performances for $\mathcal{G} = \sigma + \sum_i \mathcal{D}_i$

ratios, AU and WLU shrink both the signal and the noise, and therefore both need to be rescaled upward. In the context of Boltzmann learning algorithms, this is equivalent to lowering their temperatures.

To investigate this issue, in a second set of experiments we used different decay factors and faster schedules. In Fig. 4 we present the asymptotic world utility value for Team Game, AU, and WLU for different decay factors and starting temperatures. In the experiments plotted we used $\mathcal{G} = \sum_i w_i \mathcal{D}_i$, but we observed the same phenomena with the other $\mathcal{G}$ discussed above: the team game never outperforms AU or WLU. Moreover, if for each time-algorithm pair we used the schedule that is optimal for that pair, then there is no time t at which the performance of the team game is better than the performance of AU/WLU.

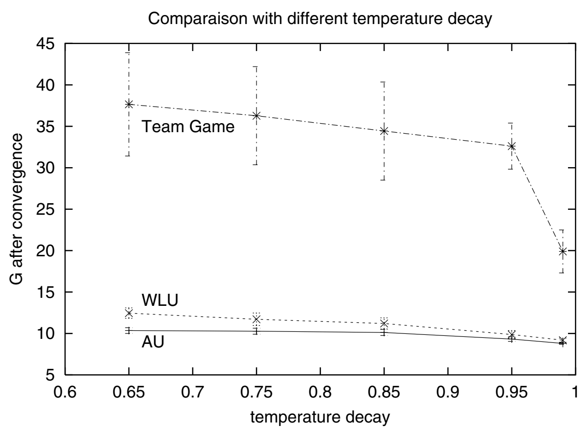


Fig. 4. Effect of decay factors ($\mathcal{G} = \sum_i w_i \mathcal{D}_i$)

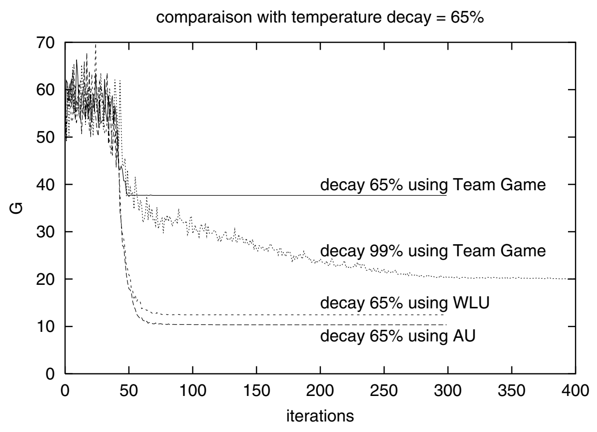


Fig. 5. Comparative performance using different decay factors (even with a fast schedule AU and WLU perform better than team game with a slow schedule). $\mathcal{G} = \sum_i w_i \mathcal{D}_i$

Fig. 5 illustrates this, showing that with a fast schedule having a decay factor of 0.65, AU and WLU converge faster than team game.

In addition to these advantages, we found that the performance with the team game is more sensitive to the change in temperature schedule, again in accord with COIN theory. Indeed, although convergence to asymptotia is faster for all agent utilities with the faster schedule (see Fig. 5), from Fig. 4 we see that both average asymptotic performance and its standard deviation are substantial for the team game for the faster schedule. In contrast, AU and WLU do not suffer as much from the switch to a faster schedule.

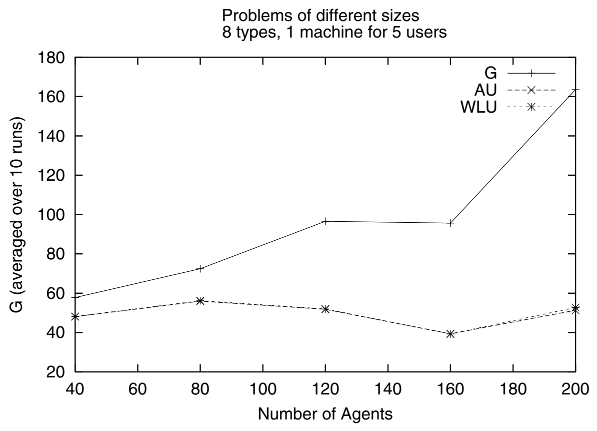


Fig. 6. Scaling properties for $\mathcal{G} = \sum_i \mathcal{D}_i$

We also investigated the scaling properties of these approaches as one increases the size of the system and found that AU and WLU scale up much better than the team game approach, again in accord with COIN theory. Intuitively, the larger the system, the worse the signal-to-noise problems with utilities like team games, and therefore the greater the gain in using a utility like AU or WLU that corrects for it. In Fig. 6, we used a fixed number of types, 8, and a fixed ratio of 1 machine for 5 users. We increased the number of users from 5 to 25. For each data point, we considered a few scenarios (different machine configurations, different loads distributed to users, etc.) and performed 10 runs for each scenario. Each point represents the average over the scenarios. The use of AU and WLU yield comparable results, and the difference in performance between them and Team Game increases with the number of agents.

Since in general one cannot compute the best possible $\mathcal{G}$ value, we cannot make absolute performance claims. Nonetheless, both in terms of performance and scaling, use of WLU or AU is clearly preferable to a team game or greedy approach, for several different choices for $\mathcal{G}$.

5 Related Work

Multiagent system researchers have studied balancing of load across shared resources with different decision-making procedures. These approaches include the following:

- Studying chaotic nature of resource loads when each agent uses a greedy selection procedures [11].
- Effect of limited knowledge on system stability [12, 13].
- Market mechanisms for optimizing quality-of-service [14].
- Using reinforcement learning to balance loads [15].

– Social dilemma problems that arise when individual agents try to greedily exploit shared resources [3, 16].
– Distinguishing easy vs. difficult resource allocation [17].

The typical assumption in most of this work is that each user has an atomic load to send to one of several equivalent resources. Our work addresses a more general scenario where a user has multiple task loads and resources are heterogeneous. So not only may the decisions of the users conflict, but decisions for different tasks taken by the same user can interfere with each other. Also, we explicitly handle the issue of user satisfaction metrics (something awkward to incorporate in load-balancing, for example) and variability in the world utility, and use the COIN procedure to ensure that the combination of individual user satisfaction metrics are optimized. The combination can include weights for different users and also, if necessary, reduce disparate levels of satisfaction from the outcome by minimizing the standard deviation of satisfaction. With the COIN procedure, all of this is done using very simple agents, in a highly parallelizable fashion, with no modeling of the underlying dynamics of the system.

6 Discussion

The COIN framework has been applied to many domains, including domains requiring sequence of actions. In these cases, the world utility function was decided first, and then the local utility function of each agent was derived from it. This paper differs in two main ways from the COIN applications studied so far. First, agents have preferences represented by a dissatisfaction function. In previous work, the individual in the system did not have any intrinsic preferences. Secondly, the world utility is based upon the preferences of users and a system administrator. The main contribution of the paper is to show that the users, in order to achieve satisficing performance of the overall system, is better off while optimizing COIN based private utilities. The results demonstrate that even in the case where the world utility function is not exogenous, the previously demonstrated superiority of COIN technique over competing approaches holds. In particular, the utilities AU and WLU outperform locally greedy approaches and more importantly, the Team Game approach, in terms of average performance, variability in performance, and robustness against changes to the parameters of the agent's learning algorithm. Moreover, these improvements grow dramatically as the system size is increased, an extremely important consideration in future applications involving scale up.

We are currently experimenting with aggregating several printing decisions under the jurisdiction of one agent. This would reduce the number of agents in the system, and possibly both speed up the convergence and improve the performance. In particular we will investigate different kind of aggregations, e.g., based on users, based on task types, crossing both, etc. There is also the interesting possibility of learning the best aggregation of decisions into individual agents.

Acknowledgements

This work has been supported in part by an NSF CAREER award IIS-9702672 and IIS-0209208 and by NASA EPSCoR Grant No. NCC5-586.

References

[1] Durfee, E.H.: Scaling up coordination strategies. IEEE Computer **34** (2001) 39–46 250

[2] Hardin, G.: The tragedy of the commons. Science **162** (1968) 1243–1248 250

[3] Tumer, K., Wolpert, D.H.: Collective intelligence and braess' paradox. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence, Menlo Park, CA, AAAI Press (2000) 104–109 250, 254, 262

[4] Wolpert, D.H., Tumer, K.: Beyond mechanism design. In et al., H.G., ed.: Proceedings of International Congress of Mathematicians, Qingdao Publishing (2002) 250

[5] Wolpert, D.H., Wheeler, K.R., Turner, K.: General principles of learning-based multi-agent systems. In: Proceedings of the Third International Conference on Autonomous Agents, New York: NY, ACM Press (1999) 77–83 250, 254

[6] Wolpert, D.H., Tumer, K.: Optimal payoff functions for members of collectives. Advances in Complex Systems **4** (2001) 265–279 250, 254, 256

[7] Tumer, K., Agogino, A.K., Wolpert, D.H.: Learning sequences of actions in collectives of autonomous agents. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, New York: NY, ACM Press (2002) 378–385 250

[8] Wolpert, D.H., Kirshner, S., Merz, C.J., Tumer, K.: Adaptivity in agent-based routing for data networks. In: Proc. of the 4th International Conference on Autonomous Agents. (2000) 396–403 250

[9] Parkes, D.C.: Iterative combinatorial auctions: Theory and practice (2001) 253

[10] Crites, R.H., Barto, A.G.: Improving elevator performance using reinforcement learning. In Touretzky, D.S., Mozer, M.C., Hasselmo, M.E., eds.: Advances in Neural Information Processing Systems - 8, MIT Press (1996) 1017–1023 253

[11] Kephart, J.O., Hogg, T., Hubermann, B.A.: Dynamics of computational ecosystems: Implications for DAI. In Huhns, M.N., Gasser, L., eds.; Distributed Artificial Intelligence. Volume 2 of Research Notes in Artificial Intelligence. Pitman (1989) 261

[12] Rustogi, S.K., Singh, M.P.: Be patient and tolerate imprecision: How autonomous agents can coordinate effectively. In: Proceedings of the International Joint Conference on Artificial Intelligence. (1999) 512–517 261

[13] Sen, S., Arora, N., Roychowdhury, S.: Using limited information to enhance group stability. International Journal of Human-Computer Studies **48** (1998) 69–82 261

[14] Yamaki, H., Yamauchi, Y., Ishida, T.: Implementation issues on market-based qos control. In: Proceedings of the Third International Conference on Multi-Agent Systems. (1998) 357–364 261

[15] Schaerf, A., Shoham, Y., Tennenholtz, M.: Adaptive load balancing: A study in multiagent learning. Journal of Artificial Intelligence Research **2** (1995) 475–500 261

[16] Glance, N.S., Hogg, T.: Dilemmas in computational societies. In: First International Conference on Multiagent Systems, Menlo Park, CA, AAAI Press/MIT Press (1995) 117–124 262

[17] van Dyke Parunak, H., Brueckner, S., Sauter, J., Savit, R.: Effort profiles in multi-agent resource allocation. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, New York: NY, ACM Press (2002) 248–255 262

A Modular Paradigm for Building Self-Organizing Peer-to-Peer Applications[⋆]

Márk Jelasity [⋆⋆], Alberto Montresor, and Ozalp Babaoglu

Department of Computer Science, University of Bologna, Italy
{jelasity,montresor,babaoglu}@cs.unibo.it

Abstract. Peer-to-peer (P2P) technology has undergone rapid growth, producing new protocols and applications, many of which enjoy considerable commercial success and academic interest. Yet, P2P applications are often based on complex protocols, whose behavior is not completely understood. We believe that in order to enable an even more widespread adoption of P2P systems in commercial and scientific applications, what is needed is a modular paradigm in which well-understood, predictable components with clean interfaces can be combined to implement arbitrarily complex functions. The goal of this paper is to promote this idea by describing our initial experiences in this direction. Our recent work has resulted in a collection of simple and robust components, which include aggregation and membership management. This paper shows how to combine them to obtain a novel load-balancing algorithm that is close to optimal with respect to load transfer. We also describe briefly our simulation environment, explicitly designed to efficiently support our modular approach to P2P protocol design.

1 Introduction

Recent years have witnessed a rapid growth in both the body of scientific knowledge on peer-to-peer (P2P) technology and its commercial applications [10]. There are several features that make P2P systems interesting for scientific research, which include their ability to exploit distributed resources, circumvent censorship and their potential for extreme scalability and robustness. As such, an important candidate consumer for this technology is the computational grid, which is supposed to enable optimal exploitation of large amounts of resources available over the Internet using a P2P approach [5].

The promises of P2P technology have already been partially fulfilled by numerous applications. Unfortunately, the underlying protocols on which they are built are often complex and unpredictable. Their behavior is not fully understood, and often, can be explained only in terms of the theory of complex networks or dynamic systems. Given the lack of traditional hard guarantees

⋆ This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923).

⋆⋆ also with RGAI, MTA SZTE Szeged, Hungary

© Springer-Verlag Berlin Heidelberg 2004

regarding expected outputs, users outside the scientific community—especially engineers and application developers—might experience difficulties in exploiting the growing body of available knowledge.

In our opinion, a more significant exploitation of P2P technology requires a *modular paradigm* where well-understood and predictable components with clean interfaces can be combined to implement arbitrarily complex functions. The goal of this paper is to report on our ideas and initial results towards this objective.

The first step in our advocated paradigm is to identify a collection of primitive components, that is, simple P2P protocols that can be used as *building blocks* for constructing more complex protocols and applications. An informal classification of these building blocks in two broad classes is possible:

Overlay protocols maintain connected communication topologies over a set of nodes. We refer to such topologies as *overlays*, as they are built over underlying networks like the Internet. An example is NEWSCAST [6], that maintains a random overlay.

Functional protocols are aimed at implementing basic functions for other components. An example for such a function is *aggregation* [17], a collective name for functions that provide global information about a distributed system. These functions include finding extremal values, computing averages and sums, counting, etc.

A modular approach offers several attractive possibilities. It allows developers to plug different components implementing a desired function into existing or new applications, being certain that the function will be performed in a predictable and dependable manner. An even more interesting possibility is to combine building blocks to form more complex applications that perform relatively sophisticated functions like file sharing or load balancing.

Building blocks must be simple and predictable, as well as extremely scalable and robust. In this way, research can focus on self-organization and other important emergent features, without being burdened by the complexity of the protocols. Our building blocks are typically no more complicated than a cellular automaton or a swarm model which makes them ideal objects for research. As a result, practical applications can also benefit from a potentially more stable foundation and predictability, a key concern in fully distributed systems.

In order to demonstrate the usefulness of our approach, we describe how to implement a fairly complex function, *load balancing*, using the two building blocks introduced above—NEWSCAST and aggregation. It turns out that the resulting protocol is close to optimal with respect to the amount of load it transfers.

The outline of the paper is as follows. In Section 2 we define our system model. Section 3 describes the basic building blocks that will be used by the load balancing example described in Section 4. In Section 5 we give a brief overview of PEERSIM, the high-level network simulator that we have specifically developed to support our modular paradigm.

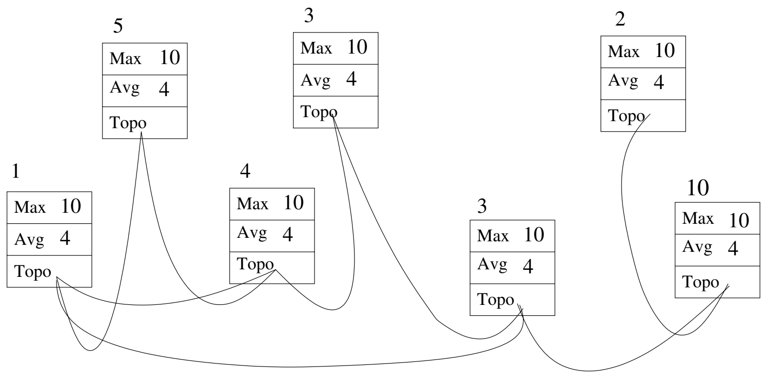


Fig. 1. A simple example network. The environment consists of a set of numeric values, one at each node. Two protocols, MAX and AVG, find the maximum and the average of these values, respectively. They are based on protocol TOPO, whose task is to maintain an overlay topology (represented by the connections between nodes)

2 System Model

Figure 1 illustrates our system model. We consider a network comprising a large collection of *nodes* that communicate through the exchange of messages and are assigned unique identifiers. The network is highly dynamic (new nodes may join and old ones can leave at any time) and subject to failures (nodes may fail and messages can be lost).

Each node runs a set of *protocols*. Protocols can be standalone applications, or may provide some service to other protocols. Each protocol instance may communicate with other protocols located at the same node (e.g., to import or export a service) and with other instances of the same protocol type located at remote nodes (e.g., to implement a function).

We assume that nodes are connected by an existing *physical network*. Even though the protocols we suggest can be deployed on arbitrary physical networks, including sensor and ad-hoc networks, in the present work we consider only fully connected networks, such as the Internet, where each node can (potentially) communicate with every other node. In this way, arbitrary overlay topologies may be constructed, and a functional protocol may deploy the most appropriate overlay for implementing its functions.

The physical network provides only the possibility of communication. To actually communicate with its peers, a node must know their identifiers. At each node, the task of an overlay protocol is to collect and maintain up-to-date identifiers in order to form a connected topology with some desired characteristics. Given the large scale and the dynamicity of our envisioned system, these collections are normally limited to a rather small subsets of the entire network.

Apart from communicating with other peers, protocols may also interact with their *environment*. Any input that originates from outside the protocol set falls into this category. The environment may include user interactions, sensor

```
do forever
  wait(T time units)
  $p \leftarrow$ GETRANDOMPEER()
  send $s$ to $p$
  $s_p \leftarrow$ receive($p$)
  $s \leftarrow$ UPDATESTATE($s, s_p$)
```

(a) active thread

```
do forever
  $s_p \leftarrow$ receive(*)
  send $s$ to sender($s_p$)
  $s \leftarrow$ UPDATESTATE($s, s_p$)
```

(b) passive thread

Fig. 2. The skeleton of a push-pull epidemic-style protocol. Notation: s is the state of this node, s_p is the state of the peer p

information, and any application-specific data such as load in a load balancing system and free space in a distributed storage system.

In our model, modularity is implemented at the level of protocols. Protocols must provide and implement well-defined interfaces, in order to allow developers to plug different implementations of the same protocol into their applications. For example, as explained in the next section, the aggregation protocols illustrated in Figure 1 make use of an overlay protocol to communicate with peer nodes so as to compute the aggregates. Any implementation of overlay can be used, as long as it provides standard interfaces and a topology with the appropriate characteristics.

3 Example Building Blocks

In this section we describe two general-purpose protocols, NEWSCAST [6] and *epidemic-style aggregation* [7], which will be used to build our load balancing scheme in Section 4. The descriptions we provide here are necessarily brief and informal; they serve simply to introduce their structure and provide some details of their characteristics. Additional information can be found in the related papers [6, 7].

Both protocols are based on the push-pull epidemic-style scheme illustrated in Figure 2. Each node executes two different threads. The *active* one periodically initiates an *information exchange* with a peer node selected randomly, by sending a message containing the local state and waiting for a response from the selected node. The *passive* one waits for messages sent by an initiator and replies with its local state. The term push-pull refers to the fact that each information exchange is performed in a symmetric manner: both peers send and receive their states.

Method UPDATESTATE builds a new local state based on the previous local state and the state received during the information exchange. The output of UPDATESTATE depends on the specific function implemented by the protocol. The local states at the two peers after an information exchange are not necessarily the same, since UPDATESTATE may be non-deterministic or may produce different outputs depending on which node is the initiator.

The period of wall clock time between two consecutive information exchanges is called the *cycle length*, and is denoted by T. Even though the system is not synchronous, it is often convenient to talk about *cycles* of the protocol, which are simply consecutive wall clock time intervals of length T counted from some convenient starting point.

3.1 Newscast

The NEWSCAST protocol [6] is an example of an overlay protocol. It maintains a random topology, which is extremely robust, and can be used as the basis for several functional protocols, including broadcast [6] and aggregation [7].

In NEWSCAST, the state of a node is given by a *partial view*, which is a fixed-size set of peer descriptors. A *peer descriptor* contains the address of the peer, along with a *timestamp* corresponding to the time when the descriptor was created. The (fixed) size of a partial view is denoted by c.

Method GETRANDOMPEER returns an address selected randomly among those in the current partial view. Method UPDATESTATE merges the partial views of the two nodes involved in an exchange and keeps the c freshest descriptors, thereby creating a new partial view. New information enters the system when a node sends its partial view to a peer. In this step, the node always inserts its own, newly created descriptor into the partial view. Old information is gradually and automatically removed from the system and gets replaced by new information (hence the name, NEWSCAST). This feature allows the protocol to "repair" the overlay topology by forgetting dead links, which by definition do not get updated because their owner is no longer active.

3.2 Properties of Newscast

In NEWSCAST, the overlay topology is defined by the content of partial views. Each descriptor in the partial view represents a directed edge in the topology, linking the node holding the descriptor to the node named in the descriptor. The basic assumption in the design of the protocol is that the set of nodes that form the network is highly dynamic, with a continuous flow of nodes joining and leaving the system. This dynamicity is reflected in the overlay topology, that is constantly changing over time, by removing obsolete information and disseminating descriptors of joining nodes.

We have shown in [6] that the resulting topology has a very low diameter and is very close to a random graph with out-degree c. According to our experimental results, choosing $c = 20$ is already sufficient for very stable and robust connectivity.

We have also shown that, within a single cycle, the number of exchanges per node can be modeled by a random variable with the distribution $1 + \text{Poisson}(1)$. In other words, on the average, there are two exchanges per cycle (one is actively initiated and the other one is passively received) and the variance of this estimate is 1. The implication of this property is that no node is more important (or overloaded) than others.

3.3 Epidemic-Style Aggregation

In the case of epidemic-style aggregation [7], the state of a node is a numeric value. In a practical setting, this value can be any attribute of the environment: storage capacity, temperature, load, etc. The task of the protocol is to calculate an aggregate value over the set of all numbers held by the nodes. Here, we will focus on the two specific cases of average and maximum. Other aggregate functions, including sum, counting, variance estimation, etc., may be easily computed using a similar scheme.

In order to function, this protocol needs an overlay protocol that provides an implementation of method GETRANDOMPEER. In the present paper, we assume that this service is provided by NEWSCAST, but any other overlay protocol could be used.

In the case of averaging, let method UPDATESTATE(a, b) return $(a+b)/2$. After one state exchange, the sum of the values maintained by the two nodes does not change, since they have just balanced their values. So the operation does not change the global average either; it only decreases the variance over all the estimates in the system. In the case of maximum, let method UPDATESTATE(a, b) return $\max(a, b)$. In this case, the global maximum value will be effectively broadcast like an epidemic.

3.4 Properties of Epidemic-Style Averaging

Maximum and averaging protocols have different mathematical properties. For maximum, existing results about epidemic-style broadcasting [3] are applicable. From now on, we focus on averaging only.

For our purposes, the most important feature will be the *convergence speed* of averaging. As mentioned above, it is guaranteed that the value at each node will converge to the true global average, as long as the underlying communication topology is connected. In [7] it was shown that if this communication topology is not only connected but also sufficiently random, the speed of convergence is exponential.

In a more precise mathematical formulation, let μ_i be the empirical mean and σ_i^2 be the empirical variance in cycle i,

$$\mu_i = \frac{1}{N}\sum_{k=1}^{N} a_{i,k}, \qquad \sigma_i^2 = \frac{1}{N-1}\sum_{k=1}^{N}(a_{i,k}-\mu_i)^2 \tag{1}$$

where $a_{i,k}$ is the value maintained at node $k = 1, \ldots N$ during cycle i and N is the number of nodes in the system. It can be shown that we have

$$E(\sigma_{i+1}^2) \approx \frac{E(\sigma_i^2)}{2\sqrt{e}}. \tag{2}$$

Simulations show that this approximation holds with high precision. From this equation, it is clear that convergence can be achieved with very high precision in only a few cycles, irrespective of the network size which confirms extreme scalability.

In addition to being fast, our aggregation protocol is also very robust. Node failures may perturb the final result, as the values stored in crashed nodes are lost; but both analytical and empirical studies have shown that this effect is generally marginal [7]. As long as the overlay network remains connected, link failures do not modify the final value, they only slow down the aggregation process.

4 Load Balancing: An Example Application

Let us define the load balancing problem, which will be our example application for illustrating the modular design paradigm. We assume that each node has a certain amount of load and that the nodes are allowed to transfer all or some portions of their load between themselves. The goal is to reach a state where each node has the same amount of load. To this end, nodes can make decisions for sending or receiving load based only on locally available information.

Without further restrictions, this problem is in fact identical to the averaging problem described in Section 3. In a more realistic setting however, each node will have a limit, or *quota*, on the amount of load it can transfer in a given cycle (where *cycle* is as defined in Section 3). In our present discussion we will denote this quota by Q and assume that it is the same for each node.

4.1 The Optimal Algorithm

For the sake of comparison, to serve as a baseline, we give theoretical bounds on the performance of any load balancing protocol that has access to global information.

Let $a_{i,1}, \dots a_{i,N}$ represent the individual loads at cycle i, where N is the total number of nodes. Let μ be the average of these individual loads over all nodes. Note that the global average does not change as a result of load transfers as long as work is "conserved" (there are no node failures). Clearly, at cycle i, the minimum number of additional cycles that are necessary to reach a perfectly balanced state is given by

$$\max_j \left\lceil \frac{|a_{i,j} - \mu|}{Q} \right\rceil \tag{3}$$

and the minimum amount of total load that needs to be transferred is given by

$$\frac{\sum_j |a_{i,j} - \mu|}{2}. \tag{4}$$

Furthermore, if in cycle i all $a_{i,j} - \mu$ $(j = 1, \dots, N)$ are divisible by Q, then the optimal number of cycles and the optimal total transfer can both be achieved by the protocol given in Figure 3.

This algorithm is expressed not as a local protocol that can be run at each node, but as a global algorithm operating directly on the list of individual loads.

```
Let $a_{i_1}, \ldots, a_{i_N}$ be the decreasing order of load values $a_1, \ldots, a_N$
$j \leftarrow 1$
while ($a_{i_j} > \mu$ and $a_{i_{N+1-j}} < \mu$)
  $a_{i_j} \leftarrow a_{i_j} - Q$
  $a_{i_{N+1-j}} \leftarrow a_{i_{N+1-j}} + Q$
  $j \leftarrow j + 1$
```

Fig. 3. One cycle of the optimal load balancing algorithm. Notation: μ is the average load in the system, N is the network size, Q is the quota

It relies on global information in two ways. First, it makes a decision based on the overall average load (μ) which is a global property and it relies on globally ordered local load information to select nodes with specific characteristics (such as over- or under-loaded) and for making sure the quota is never exceeded.

It is easy to see that the total load transfered is optimal, since the load at each node either increases monotonically or decreases monotonically, and when the exact global average is reached, all communication stops. In other words, it is impossible to reach the balanced state with any less load transfered.

The algorithm also achieves the lower bound given in (3) for the number of cycles necessary for perfect balance. First, observe that during all transfers exactly Q amount of load is moved. This means that the property that all $a_{i,j} - \mu$ ($j = 1, \ldots, N$) are divisible by Q holds for all cycles, throughout the execution of the algorithm. Now, we only have to show that if $\max_j |a_{i,j} - \mu| = kQ \geq 0$ then

$$\max_j |a_{i,j} - \mu| - \max_j |a_{i+1,j} - \mu| = Q. \tag{5}$$

To see this, define $J = \{j^* | \max_j |a_{i,j} - \mu| = |a_{i,j^*} - \mu|\}$ as the indices which belong to nodes that are maximally distant from the average. We have to show that for all nodes in J, a different node can be assigned that is on the other side of the average. We can assume without the loss of generality that the load at all nodes in J is larger than the average because (i) if it is smaller, the reasoning is identical and (ii) if over- and under-loaded nodes are mixed, we can pair them with each other until only over- or under-loaded nodes remain in J. But then it is impossible that the nodes in J cannot be assigned different pairs because (using the definition of J and the assumption that all nodes in J are overloaded) the number of under-loaded nodes has to be at least as large as the size of J. But then all the maximally distant nodes got their load difference reduced by exactly Q, which proves (5).

Motivated by this result, in the following we assume that (a) the initial load at each node is an integer value, (b) the average is also an integer and (c) we are allowed to transfer at most one unit of load at a time. This setting satisfies the assumptions of the above results and serves only as a tool for simplifying and focusing our discussion.

```
do forever
  $q \leftarrow Q$
  wait(T time units)
  $\mu \leftarrow$ GETAVERAGELOAD()
  if ($q = 0$) continue
  if ($|a - \mu| < Q$) FREEZE()
  if ($a < \mu$)
    $p \leftarrow$ GETOVERLOADEDPEER($q, \mu$)
    if ($p \neq$ null) TRANSFERFROM($p, q$)
  else
    $p \leftarrow$ GETUNDERLOADEDPEER($q, \mu$)
    if ($p \neq$ null) TRANSFERTO($p, q$)
```

(a) active thread

```
GETOVERLOADEDPEER($q, \mu$)
  $(p_1, \ldots, p_c) \leftarrow$ GETNEIGHBORS()
  Let $p_{i_1}.a, \ldots, p_{i_c}.a$ be the decreasing
    order of neighbor load values $p_1.a, \ldots, p_c.a$
  for $j = 1$ to $c$
    if ($p_{i_j}.a > \mu$ and $p_{i_j}.q \geq q$)
      return $p_{i_j}$
  return null

GETUNDERLOADEDPEER($q, \mu$)
// Defined analogously
```

(b) peer selection

Fig. 4. A modular load balancing protocol. Notations: a is the current load, Q is the total quota, q is the residual quota and c is the number of peers in the partial view as determined by the overlay protocol

4.2 A Modular Load Balancing Protocol

Based on the observations about the optimal load balancing algorithm, we propose a protocol that is based purely on local knowledge, but that approximates the optimal protocol extremely well, as we show in Section 4.4.

Figure 4 illustrates the protocol we propose. The basic idea is that each node periodically attempts to find a peer which is on the "other side" of the global average and has sufficient residual quota. If such a peer can be found, load transfer is performed.

The approximation of the global average is obtained using method GETAVERAGELOAD, and the peer information is obtained using method GETNEIGHBORS. These methods can be implemented by any appropriate component for average calculation and for topology management.

We assume that in each cycle, each node has access to the current load and residual quota of its peers. This latter value is represented by local variable q at each node, which is initialized to Q at the beginning of each cycle and is updated by decrementing it by the actual transfered load. This information can be obtained by simply asking for it directly from the peers. This does not introduce significant overhead as we assume that the load transfer itself is many orders of magnitude more expensive. Furthermore, as we mentioned earlier, the number of peers is typically small ($c = 20$ is typical).

Note that once the local load at a node is equal to the global average, the node can be excluded from future considerations for load balancing since it will never be selected for transfers. By excluding these "balanced" nodes, we can devote more attention to those nodes that can benefit from further transfers. The protocol of Figure 4 implements this optimization through the method FREEZE. When a node executes this method, it starts to play "dead" towards the overlay protocol. As a result, the node will be removed from the communication topology

```
do forever
  $q \leftarrow Q$
  wait(T time units)
  if ($q = 0$) continue
  $p \leftarrow$ GETPEER($q, a$)
  if ($p.a < a$) TRANSFERTO($p, q$)
  else TRANSFERFROM($p, q$)
```

(a) active thread

```
GETPEER($q, a$)
  $(p_1, \ldots, p_c) \leftarrow$ getNeighbors()
  Let $p_{i_1}.a, \ldots, p_{i_c}.a$ be the decreasing
    order of neighbor load values $p_1.a, \ldots, p_c.a$
    according to the ordering defined by
    $|a - p_1.a|, \ldots, |a - p_c.a|$
  for $j = 1$ to $c$
    if ($p_{i_j}.q \geq q$) return $p_{i_j}$
  return null
```

(b) peer selection

Fig. 5. The basic load balancing protocol. Notations: a is the current load, Q is the total quota, q is the residual quota and c is the number of peers in the partial view as determined by the overlay protocol

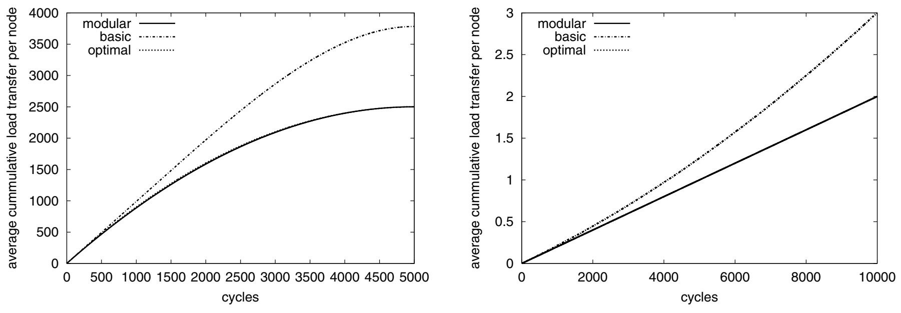
and the remaining nodes (those that have not yet reached the average load) will meet each other with higher probability. In other words, peer selection can be more efficient in the final phases of the execution of the balancing protocol when most nodes already have reached the average load. Although the optimization will result in a communication topology that is partitioned, the problem can easily be solved by adding another overlay component that does not take part in load balancing and is responsible only for maintaining a connected network. Note also that the averaging component uses the same overlay component that is used by the load balancing protocol.

A key feature of the averaging and overlay protocols is that they are potentially significantly faster than any load balancing protocol. If the quota is significantly smaller than the variance of the initial load distribution, then reaching the final balanced state can take arbitrarily long (see Equation (3)). On the other hand, averaging converges exponentially fast as defined by Equation (2). This fact makes it possible for load balancing to use the approximation of the global average as if it were supplied by an oracle with access to global information. This scenario where two (or more) protocols operate at significantly different time scales to solve a given problem is encountered also in nature and may characterize an interesting general technique that is applicable to a larger class of problems.

4.3 A Basic Load Balancing Protocol

In order to illustrate the effectiveness of using the averaging component, we suggest a protocol which does not rely on the average approximation. The protocol is shown in Figure 5.

This protocol attempts to replace the average approximation by heuristics. In particular, instead of choosing a peer from the other side of the average, each node picks the peer which has a maximally different load (larger or smaller) from the local load. The step which cannot be replaced however is the FREEZE opera-



(a) linear load distribution (b) peak load distribution

Fig. 6. Cumulative average load transferred by a node until a given cycle in a network of size 10^4. The curves corresponding to the optimal algorithm and the modular protocol overlap completely and appear as a single (lower) curve. The final point in both graphs (5000 and 10000 cycles, respectively) correspond to a state of perfect balance reached by all three protocols

tion. Performing that operation depends crucially on knowing the global average load in the system.

4.4 Empirical Results

Empirical studies have been performed using the simulator described in Section 5. We implemented the three protocols described above: the optimal algorithm, the modular protocol that is based on the averaging protocol and NEWSCAST and the basic protocol that has no access to global average load. As components, the methods of Figure 4 were instantiated with the aggregation protocol of Section 3 for averaging and NEWSCAST for the overlay.

In all our experiments, the network size was fixed at $N = 10^4$ and the partial view size was $c = 40$. We examined two different initial load distributions: linear and peak. In the case of linear distribution, the initial load of node i ($i = 0, \ldots, N$) was set to exactly $i - 1$ units. In the case of peak distribution, the load of exactly one node was set to 10^4 units while the rest of the nodes had no initial load. The total quota for load transfer in each cycle was set to one load unit ($Q = 1$).

During the experiments the variance of the local load over the entire network was recorded along with the amount of load that was transfered during each cycle. We do not show the data on variance—which would give information about the speed of reaching the balanced state—because all three protocols have identical (i.e., optimal) convergence performance for both initial distributions.

Figure 6 presents results for total load transfered during the execution of the three solutions. Each curve corresponds to a single execution of a protocol, as the variance of the results over independent runs is diminishing. As can be seen from the figures, the load transfered by the modular protocol is indistinguishable from the amount that is optimal for perfect balancing in the system.

5 A Dedicated Simulator: PeerSim

Evaluating the performance of P2P protocols is a complex task. One of the main reasons for their success, i.e. the extremely large scale that they may reach, is also one of the major obstacles for their evaluation. P2P networks that incorporate hundreds of thousands of nodes are not uncommon; at the time of writing, more than 5 million hosts are connected to the Kazaa/Fasttrack network. Another source of complexity is the high dynamicity of such systems: P2P networks are in a continuous state of flux, with new nodes joining and existing nodes leaving or crashing.

Evaluating a protocol in real settings, especially in the first phases of its design, is clearly not feasible. Even the larger distributed testbeds for deploying planetary-scale network services [12] do not include more than 220 nodes, a tiny figure compared to the size of modern P2P applications. Furthermore, nodes in such testbeds are not characterized by the same degree of dynamicity that is typical of P2P nodes.

For some protocols, an analytical evaluation is possible. For example, simple epidemic-style protocols (such our average aggregation mechanism) may analyzed mathematically due to their simplicity and their inherent probabilistic nature. Nevertheless, simulation is still an invaluable tool for understanding the behavior of complex protocols or validating theoretical results.

The results illustrated in the previous section have been obtained using our simulator called PeerSim, developed by our group and specialized for supporting the simulation of P2P protocols based on the modular paradigm pursued in this paper. PeerSim complements our model by enabling developers to experiment with protocols and their composition.

5.1 Design Objectives for PeerSim

A simulator for P2P systems may have very different objectives from general-purpose networks simulators [11]:

- *Extreme Scalability.* Simulated networks may be composed of millions of nodes. This may be obtained only if a careful design of the memory layout of the simulator is performed. Being able to store data for a large number of nodes, however, is not the only requirement for large-scale simulations; the simulation engine must be optimized as well, trying to reduce, whenever possible, any form of overhead.
- *Support for Dynamicity.* The simulator must be capable to deal with nodes that join and leave the network, either definitively or temporarily. This has some implications on memory management in the simulator, requiring mechanisms for removing nodes that are not useful any more.

In addition to these requirements, the modular approach we are pursuing in this paper must be reflected in the architecture of the simulation environment as well. The idea is to provide a composition mechanism that enables the construction of simulations as collections of components. Every component of the simulation (for example, protocols or the environment) must be easily replaceable through simple configuration files. The flexibility offered by this mechanism should enable developers to re-implement, when needed, every component of the system, with the freedom of re-using existing components for fast prototyping.

Some of these goals may appear contradictory. For example, a modular approach may introduce overhead that limits overall performance of the simulator, or smart but large data structures may improve speed, but they may also limit the scalability of the simulator. A careful design is needed trying to obtain the best equilibrium.

5.2 Simplifying Assumptions

The strong scalability requirements outlined in the previous section force us to introduce several simplifying assumptions. For example, low-level details, such as the overhead associated to the communication stack (e.g. TCP or UDP) cannot be taken into consideration because simulating the underlying protocols requires a lot of additional memory and time, a price that cannot be easily paid when nodes are in the range of millions.

However, in many cases the properties of certain P2P protocols enable us to apply not only these assumptions, but also additional ones without sacrificing much of the realism of the simulations. For example, let us consider the membership and aggregation protocols presented in Section 3. In each cycle, every node sends and receives two messages on average. In both protocols, messages are small: a few hundred bytes for NEWSCAST, and a few bytes for average aggregation. Given the fast convergence of these protocols, in a real implementation the cycle length can be chosen large enough to guarantee that messages will arrive before the start of the next cycle. (For example, choosing a cycle length of five seconds and performing 20 cycles (sufficient to obtain a variance reduction of 10^{-9}), less than two minutes are needed to obtain the average, independently of the size of the network.)

As a result of these properties even latency and bandwidth may be dropped from our models, without rendering the simulations unrealistic. For these reasons, the simulation model that is adopted by PEERSIM ignores concurrency and in fact it is very similar to a cellular automaton model. The model is based on the concept of cycle. In each cycle all nodes have a chance to perform a basic operation based on their current state and possible communication with their current neighboring nodes, where neighborhood relation is defined by an overlay protocol or a fixed communication topology.

5.3 The Peersim Architecture

The architecture of PeerSim is illustrated in Figure 7. As described above, PeerSim has been designed to be highly modular and configurable, without incurring in excessive overhead both in terms of memory and time.

The *configuration manager* is the main component. Its task is to read configuration files and command-line parameters, and compose a simulation starting from the components listed in the configuration. The configuration manager is the only fixed module of a simulation; every other component is interchangeable, to allow developers to write their customized and optimized version when needed. The configuration mechanism is currently based on Java property files, that are collections of pairs associating a property name to a property value. Each configuration file is substantially a list of associations between component identifiers and the name of the Java class implementing the particular protocol. After the instantiation, each component is responsible for reading the additional parameters needed by its implementation. For example, a membership component may read the configured maximum size of its partial view.

Following the definitions provided in Section 2, the simulated network is composed of a collection of *nodes*, each of them may host one or more *protocols*. Communication between protocol instances belonging to the same node are based on method invocations: in order to provide a service, each protocol must implement a well-defined interface. For example, protocols that maintain an overlay topology must implement the `Linkable` interface, that enables higher-level services to obtain information about the neighbors known to that node. Protocols implementing aggregation must provide an interface for setting the local value and obtaining the aggregated one.

The interaction between the environment and the protocols is represented by *Dynamics*, that are executed periodically by the simulation engine, and may interact with the simulated systems at different levels; for example, they may modify the network composition, either by adding new nodes or by destroying existing ones; or, they may act at the level of protocols, for example modifying an overlay topology or changing the aggregated value.

Observers play the role of global observation points from which it is possible to analyze the network, the nodes composing it and the state of the protocols included on them, in order to collect statistics about the behavior of the system as a whole. Observers, like dynamics, are executed periodically. Observers may be highly customized for a particular protocol (for example, to report the variance reduction rate in an aggregation protocol), or may be more general (for example, to analyze graph-theoretical properties of maintained topologies, like diameter, clustering, etc.).

Protocols, dynamics and observers give designers of P2P protocols complete freedom to simulate whatever system they want, at the desired level of accuracy; yet, the presence of a growing library of pre-built components enable the construction of fast prototypes.

The *simulation engine* is the module that will actually perform the simulation; its task is to orchestrate the execution of the different components loaded

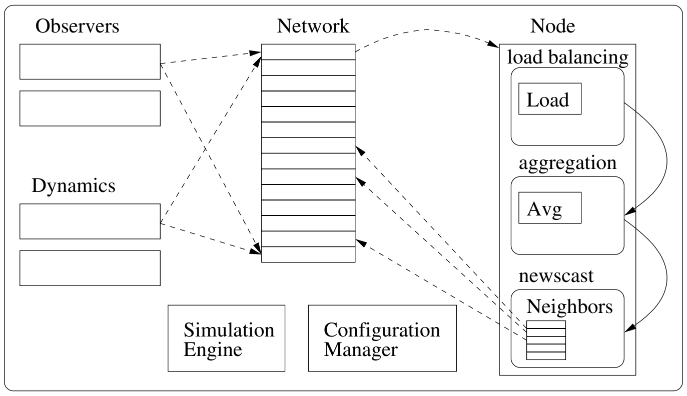


Fig. 7. Architecture of the PeerSim simulator. A simulation is composed of a set of nodes, observers and dynamics objects. Nodes are composed of a set of protocols. The composition of a single node is shown, with the protocols used to implement load balancing

in the system. As described in the previous section, the engine adopts a time-stepped simulation model instead of more complex and expensive event-based architecture. At each time step, all nodes in the system are selected in a random order, and a callback method is invoked on each of the protocols included in that node. In this way, all protocol instances get a chance to execute at each cycle. Dynamics and observers are executed periodically; each of them is associated with a *scheduler* object, that based on the information stored in the configuration file decides when and how frequently the particular dynamics or observer must be executed.

Communication between nodes is left to the designer of protocols; in the algorithms developed so far, protocol instances invoke methods on each other, in order to reduce the overhead associated to the creation, the enqueuing and the garbage collection of messages. This is compatible with the kind of protocols simulated so far. Using this approach, nodes are responsible to check whether a particular node cannot be reached, either due to a node crash or a communication failure; if so, the method invocation should be discarded.

6 Related Work

Another well-known functional building block is a *distributed hash table* (DHT), which is an abstraction of a distributed data structure that maintains associations between keys and nodes in the network. There have been proposals for applying DHTs as abstract building blocks to construct more complex applications [2] including event notification systems [14] and distributed storage systems [4]. Yet, DHTs themselves are often complex and in our conceptual framework, we are looking for possibilities for decomposing them into smaller components [18].

The load balancing problem, which we have used as an example to demonstrate the usefulness of our approach, is one of the oldest problem in distributed systems. Past research has proposed several different load balancing strategies and has evaluated their performance on both distributed systems and multiprocessors. In these studies, the topologies considered are either fixed, structured graphs (such as trees, rings, stars, multi-dimensional hypercubes, etc.) [9], or complete graphs, where each node knows the identifier of every other node [1]. As such, these results are not easily applicable to P2P networks. Due to the high dynamicity and large-scale of these networks, constructing and maintaining topologies as overlay networks is a complex task, and for the same reasons, complete overlays are not possible. Furthermore, and more importantly, these structured topologies do not exploit the possibility of constructing close-to-optimal overlay networks, as we did in this paper.

More recently, Rao et al. have presented a set of algorithms for solving the load balancing problem in DHT networks [13]. One of these algorithms, called one-to-many, is similar to our approach: they assume the knowledge of a known target for the average load , and each over-loaded node selects the most under-loaded nodes among a subset. Their algorithm, however, does not explain exactly how the average load may be obtained, and does not exploit the possibility of progressively reducing the overlay network to those nodes that need to be balanced.

Surveying existing literature on P2P simulation, the picture that is obtained is highly fragmented; most current P2P projects base their results on home-grown simulators that have been tuned for the particular protocols employed by the target systems. Very few papers [19] make use of general-purpose, detailed network simulators such as NS2 [11]. This choice is due to the high costs associated with packet-level simulations that pose very severe limits to scalability.

Despite this fragmentation, several interesting simulation environments have appeared recently [15, 16, 8]. Most of them are limited to file-sharing applications, as they incorporate notions of documents, keywords, etc. The most promising one is NeuroGrid [8], that has been designed with extensibility in mind. Yet, none of them approach the scalability that can be obtained by PeerSim.

7 Conclusions and Future Work

In this paper we have presented our initial results towards a modular paradigm for designing complex, self-organizing protocols. A load balancing protocol was presented to confirm the validity of the idea that simple components can be combined to implement more complex functions. We demonstrated the utility of applying an averaging component along with an overlay component to obtain a performance with respect to the amount of transfered load that is indistinguishable from the optimal case.

Naturally, our present and future work focuses on developing this paradigm further by extending it with more components and applications. It is interesting to consider the similarities of this approach to object-oriented design and programming. In a sense, at least in the design phase, the designer can treat these building blocks as objects that maintain a state and that have an interface for modifying or monitoring that state or for performing functions. Some of the usual relations such as dependence or inheritance can also be applied. Our future work includes developing this analogy further.

References

[1] A. Barak and A. Shiloh. A Distributed Load Balancing Policy for a Multicomputer. *Software Practice and Experience*, 15(9):901–913, Sept. 1985. 280

[2] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, USA, Feb. 2003. 279

[3] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Management. In *Proc. of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, Vancouver, Aug. 1987. ACM. 270

[4] P. Druschel and A. Rowstron. PAST: A Persistent Anonymous Store. In *HotOS VIII*, May 2001. 279

[5] I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, USA, Feb. 2003. 265

[6] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast Computing. submitted for publication. 266, 268, 269

[7] M. Jelasity and A. Montresor. Epidemic-Style Proactive Aggregation in Large Overlay Networks, 2004. To appear. 268, 269, 270, 271

[8] S. Joseph. An Extendible Open-Source Simulator. *P2P Journal*, Nov. 2003. 280

[9] P. Kok, K. Loh, W. J. Hsu, C. Wentong, and N. Sriskanthan. How Network Topology Affects Dynamic Load Balancing. *IEEE Parallel & Distributed Technology*, 4(3), Sept. 1996. 280

[10] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Labs, Palo Alto, 2002. 265

[11] Network Simulator 2. `http://www.isi.edu/nsnam/ns/`. 276, 280

[12] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. of the First ACM Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, Oct. 2002. 276

[13] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Structured P2P Systems. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, USA, Feb. 2003. 280

[14] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The Design of a Large-scale Event Notification Infrastructure. In J. Crowcroft and M. Hofmann, editors, *Networked Group Communication, Third International Workshop (NGC'2001)*, volume 2233 of *Lecture Notes in Computer Science*, pages 30–43, Nov. 2001. 279

[15] M. Schlosser and S. Kamvar. Simulating a file-sharing p2p network. In *Proc. of the First Workshop on Semantics in P2P and Grid Computing*, 2002. 280

[16] N. S. Ting and R. Deters. A P2P Network Simulator. In *Proc. of the 3rd IEEE International Conference on Peer-to-Peer Computing (P2P'03)*, Linkoping, Sweden, Sept. 2003. 280

[17] R. van Renesse. The Importance of Aggregation. In A. Schiper, A. A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, editors, *Future Directions in Distributed Computing*, number 2584 in Lecture Notes in Computer Science, pages 87–92. Springer, 2003. 266

[18] S. Voulgaris and M. van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer networks. In *Proceedings of the 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, (DSOM 2003)*, number 2867 in Lecture Notes in Computer Science. Springer, 2003. 279

[19] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *To appear in IEEE Journal on Selected Areas in Communications*, 2003. 280

Tools for Self-Organizing Applications Engineering

Carole Bernon[1], Valérie Camps[2], Marie-Pierre Gleizes[1], and Gauthier Picard[1]

[1] IRIT – Université Paul Sabatier
118, Route de Narbonne, 31062 Toulouse, Cedex 4, France
+33 561 558 343
{bernon,gleizes,picard}@irit.fr
http://www.irit.fr/SMAC
[2] L3I – Université La Rochelle
Avenue M. Crépeau, 17042 La Rochelle, Cedex 1, France
+33 546 458 732
vcamps@univ-lr.fr

Abstract. Nowadays, the applications to be realized are more often distributed, complex and open, e.g., applications on the Internet: information search, brokerage, e-commerce, e-business… Therefore, designers cannot implement a global control and list all situations such systems have to be faced with. ADELFE[1] methodology was proposed to develop this kind of software. It is based on the AMAS theory (Adaptive Multi-Agent Systems) and the emergence concept. This theory gives local agent design criteria so as to enable the emergence of an organization within the system and thus, of the global function of the system. This paper focuses on three tools of the methodology associated with the process and the UML/AUML notations. The first tool is based on the commercial software OpenTool, enriched to take into account adaptive multi-agent system development. The second tool is a support decision tool to help designers to decide if the AMAS theory is relevant for the current system to design. The last tool is an interactive tool which supports the process and helps designers to follow the process and to execute associated tasks. The use of each tool is illustrated by ETTO (Emergent TimeTabling Organization) application.

1 Introduction

Through several examples, Heylighen defines self-organization as "the spontaneous emergence of global coherence out of the local interactions between initially independent components" [13]. Because we live in a complex world, self-organizing systems are widespread in many different domains (physics, biology, economics, etc.). Nowadays, applications in computer science are becoming more and more open and complex (e.g., search for information on the Internet…). Thus, having a global view or control on such systems is not realistic. For several years, we studied self-

[1] ADELFE is a French RNTL-funded project which partners are: Artal (http://www.artal.fr) and TNI-Valiosys (http://www.tni-valiosys.com) from the industry and IRIT (http://www.irit.fr) and L3I (http://www-l3i.univ-lr.fr) from the academia. This project was started in December 2000 and will end in September 2003.

G. Di Marzo Serugendo et al. (Eds.): AAMAS 2003 Ws ESOA, LNAI 2977, pp. 283-298, 2004.
© Springer-Verlag Berlin Heidelberg 2004

organization as a possible way to successfully overcome this complexity and this openness [11]. That led us to propose a theory called "AMAS[2] theory" in which cooperation is the means by which the system self-organizes and then adjusts to changes of the environment [10]. Interactions between the internal parts (agents) of the system depend on agents' local view and on their ability to "cooperate" with each other. Changing these local interactions reorganizes the system and thus changes its global behavior.

Because all the systems do not need self-organization to adapt, we are more specifically interested into those that are open, complex, incompletely specified, with a dynamical environment and without an a priori known algorithm to find a solution. The AMAS theory has been successfully applied to several such problems (e.g., equation solving or adaptive routing of the traffic in a telephone network) and several projects have been developed:

- ABROSE (Agent based BROkerage SErvices in electronic commerce) is an electronic commerce tool for mediation of services [9];
- ANTS has studied the cooperative self-organization of ant-like robots [23];
- FORSIC (FOrmation et Recherche en Sciences de l'Information et de la Communication) provides a cooperative tool to manage the knowledge needed in assistance in information retrieval formation [16];
- STAFF (Software Tool for Adaptive Flood Forecast) is an adaptive flood forecast that is now used in an industrial environment [8].

Nevertheless, to promote the use of self-organizing systems based on the AMAS theory, tools are needed. They are required both to reuse our know-how and to guide an engineer during an application design. As a result, we are currently working on ADELFE, a toolkit to develop software with emergent functionality, which consists of an agent-oriented methodology and a library of components that can be used to make the application development easier. The originality of the ADELFE methodology resides in the fact that some tools are provided to guide a designer when applying its process and using its notations.

So, this paper aims to show how a system can be engineered by using these tools: an interactive tool to help designers to follow the process and OpenTool, a piece of software that supports the UML and AUML [17] notations. An application named ETTO for "Emergent TimeTable Organization" will be used as an example to clarify some specific points. Building a university timetable by taking into account several actors and their constraints is a complex problem for which there is no admitted algorithm. These characteristics are required when applying the AMAS theory.

This paper is then structured as follow: some requirements about the ETTO problem are presented in the second section before giving an overview of the ADELFE methodology in the third one. Tools that are provided with ADELFE are then presented in the next section in order to show how they can be used to engineer a self-organizing system in which the different parts are changing their interactions using cooperation.

[2] AMAS stands for Adaptive Multi-Agent Systems. See http://www.irit.fr/SMAC.

2 ETTO Requirements

In order to show how a self-organizing application can be developed using the tools linked with ADELFE, the next sections will refer to the ETTO application. Description and design of the system related to ETTO are not the main objective of this article and more information is available in [1].

The chosen problem is a classical course timetabling one in which timeslots and locations must be assigned to teachers and students groups in order to let them meet during lectures. The involved actors (teachers, students groups and rooms) have a certain number of constraints that must be fulfilled. Furthermore, we are interested in the case in which constraints may dynamically evolve during the search for a solution without interrupting it and restarting from scratch. Usually, solutions to such a problem can be found using different techniques like constraint-based ones or metaheuristics techniques (simulated annealing, tabu search, graph coloring□) and more recently, neural networks, evolutionary or ant algorithms [2,22]. However, no real solving technique exists when the constraints can dynamically evolve and when the system needs to adapt. Because this problem is not a simulation one, because it is actually complex when handmade or not (it belongs to the NP-complete class of problems) and has no universal solution, we do think that it represents the right example to apply the AMAS theory. The aim is to make a solution emerge, at the macro-level of the built MAS, from the interactions of independent parts at the micro-level.

General requirements for this ETTO problem are the following. As mentioned before, stakeholders are teachers, students groups and lecture rooms. Every actor individually owns some constraints that must be (best) fulfilled. A teacher has some constraints about his availabilities (the days or the timeslots during which he can teach□), his capabilities (the topics he can lecture on□) and the needs he possesses about particular pedagogic equipments (overhead projectors, video projectors, a definite lecture room for a practical work□). A students group must take a particular teaching made up of a certain number of timeslots for a certain number of teaching topics (X timeslots for a topic 1, Y timeslots for a topic 2□). A lecture room is equipped or not with specific equipments (an overhead projector, a video projector, equipment for practical works□) and can be occupied or not (during a given timeslot, on a certain day□ .).

Before presenting the tools that will be used to design the system able to solve this problem, let us describe the characteristics of the agent-oriented methodology we are going to apply.

3 ADELFE Presentation

ADELFE is a toolkit to develop software with emergent functionality. This toolkit provides several tools: a methodology to help developers to design multi-agent systems and a library of components to re-use existing expertise. The methodology provides a process, a notation and a tool to guide designers to use the given process.

Because few of the existing agent-oriented methodologies are able to deal with dynamics or evolutionary environments, the ADELFE methodology is devoted to designing applications with such characteristics. Contrary to Gaia [24] or Tropos [3], ADELFE is not a general methodology; it concerns applications in which self-organization makes the solution emerge from the interactions of its parts. It also gives some hints to designers to determine if using the AMAS theory is relevant to build the current application.

To be as close as possible of standards and tools already known and used by engineers, the ADELFE methodology [1] is based on object-oriented methodologies, follows the RUP (Rational Unified Process) [12] and uses the UML (Unified Modelling Language) and AUML (Agent-UML) [17] notations. ADELFE covers the entire process of software engineering; during each workflow, designers can backtrack to previous results to update or complete them. OMGs SPEM (Software Process Engineering Metamodel) [18] has been used to express this process. Thus, the first four work definitions of this process can be listed as done in the Fig. 1, using the SPEM vocabulary (work definitions (WD_i), activities (A_j), steps (S_k),

As this methodology concerns only applications designed following the AMAS theory, some activities or steps had been added to the RUP in order to be specific to adaptive multi-agent systems. Furthermore, this methodology is something more than a one-moremethodology that is why, in the next section, we explain how its special features are distinguishing it from some of the already existing agent-oriented methodologies. Among the different activities or steps that are listed in the Fig. 1, some are marked with a bold font to show that they are specific to adaptive multi-agent systems. In this section, these add-ons are briefly explained and our approach is compared to some existing methodologies.

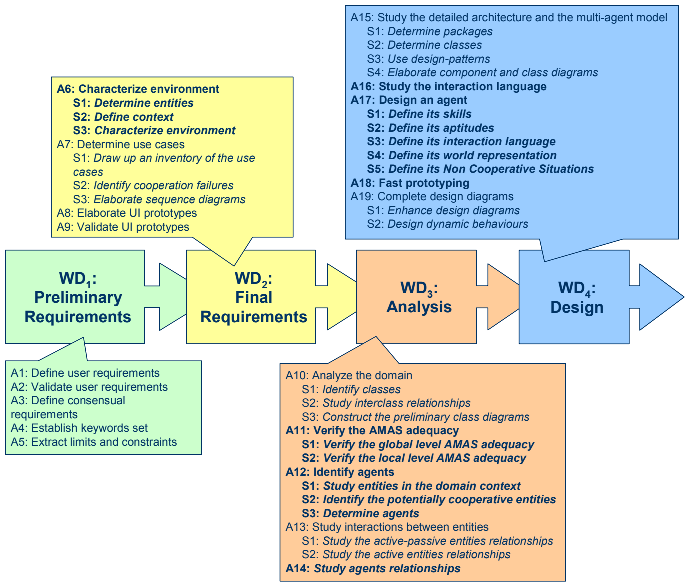

A15: Study the detailed architecture and the multi-agent model
S1: Determine packages
S2: Determine classes
S3: Use design-patterns
S4: Elaborate component and class diagrams
A16: Study the interaction language
A17: Design an agent
S1: Define its skills
S2: Define its aptitudes
S3: Define its interaction language
S4: Define its world representation
S5: Define its Non Cooperative Situations
A18: Fast prototyping
A19: Complete design diagrams
S1: Enhance design diagrams
S2: Design dynamic behaviours
A6: Characterize environment
S1: Determine entities
S2: Define context
S3: Characterize environment
A7: Determine use cases
S1: Draw up an inventory of the use cases
S2: Identify cooperation failures
S3: Elaborate sequence diagrams
A8: Elaborate UI prototypes
A9: Validate UI prototypes
WD1: Preliminary Requirements
WD2: Final Requirements
WD3: Analysis
WD4: Design
A1: Define user requirements
A2: Validate user requirements
A3: Define consensual requirements
A4: Establish keywords set
A5: Extract limits and constraints
A10: Analyze the domain
S1: Identify classes
S2: Study interclass relationships
S3: Construct the preliminary class diagrams
A11: Verify the AMAS adequacy
S1: Verify the global level AMAS adequacy
S2: Verify the local level AMAS adequacy
A12: Identify agents
S1: Study entities in the domain context
S2: Identify the potentially cooperative entities
S3: Determine agents
A13: Study interactions between entities
S1: Study the active-passive entities relationships
S2: Study the active entities relationships
A14: Study agents relationships

Fig. 1. ADELFE process is described in three levels: work definitions (WDi), activities (Ai) and steps (Si)

Preliminary and Final Requirements. Expression of requirements is a fundamental activity in software engineering and is taken into account in ADELFE as well as in MESSAGE [7], PASSI [4] or Tropos which are methodologies that cover the entire process of software engineering. In self-organizing systems as viewed by the AMAS theory, the adaptation process starts from the interactions between the system and its environment. Therefore, after having established an agreement on the preliminary requirements, designers have to model this environment. Three steps, added during the elaboration of the final requirements, enable to draw up this model: determining the entities that are involved in the system, defining the context and characterizing the environment (A6). To characterize the environment, designers must think about the environment of the system to build in terms of being accessible or not, deterministic or not, dynamic or static and discrete or continuous. These terms have been reused from [21] and represent a help to later determine if the AMAS technology is needed or not.

Analysis. In the analysis work definition, two activities have been added to the RUP: the adequacy of the AMAS theory (A11) and the identification of agents (A12).

Knowing very early if the system to develop justifies some investment in a new methodology or technique is very important. Because an adaptive MAS is not a technical solution for every application, ADELFE is the only methodology that provides a tool to help designers to decide if this theory is adequate to implement a given application. Depending on the answers given by designers concerning a certain number of criteria (see section 5), this tool estimates if the AMAS technology is useful; if that is not the case, it could suggest to use another technology.

ADELFE also focuses on the agent identification, like in AAII [14], MESSAGE and PASSI, because it does not assume that all the entities defined during the final requirements are agents. In some methodologies –e.g. Gaia or Tropos– the question is not even asked; designers already know that every entity is viewed as an agent. However, in ADELFE, the notion of agent is restrictive because we are only interested in finding what we call “cooperative agents”. A cooperative agent ignores the global function of the system; it only pursues an individual objective and tries to be permanently cooperative with other agents involved in the system. The global function of the system is emerging (from the agent level to the multi-agent level) thanks to these cooperative interactions between agents. An agent can also detect Non Cooperative Situations (NCS) that are situations it judges being harmful for the interactions it possesses with others, such as lack of understanding, ambiguity, uselessness… This does not mean that an agent is altruistic or always helping other agents but it is just trying to have useful (from its point of view) interactions with others. Thus, facing with a NCS, an agent always acts to come back to a cooperative state. In fact, the behavior of the collective compels the behavior of an agent. In ADELFE, designers are given some guidelines: an entity can become an agent if it can be faced with “cooperation failures” and may have evolutionary representations about itself, other entities or about its environment and/or may have evolutionary skills.

Design. During the design stage, three activities are added to the RUP: study of the interaction languages (A15), building of the agent model (A16) and fast prototyping (A17).

Some methodologies are devoted to a specific kind of agents, such as the AAII methodology with BDI agents. ADELFE is dedicated to cooperative agents. It provides an agent model and designers must then describe the architecture of a cooperative agent by giving the components realizing its behavior: skills, aptitudes, interaction language, world representation and the NCS this agent can encounter [1]. The global function of a self-organizing system is not coded; designers only have to implement the local behaviors of the parts composing the system. In an AMAS, the NCS model is essential because agents are changing their interactions to try to deal with cooperation failures. The NCS of an agent are depending on the application and must be described by designers. To help them, ADELFE provides generic cooperation failures such as incomprehension, ambiguity, uselessness or conflict and tables with several fields to fill up. These fields concern the name of a NCS, its generic type, the state in which the agent must be to detect it, the conditions of its detection and what actions the agent must perform to deal with it.

By following the previously described process, three tools are available. First, the OpenTool graphical tool allows designers to realize the different UML/AUML diagrams and some validations of the software specification and design. Then, the adequacy AMAS tool helps designers in choosing AMAS technology to design systems. Finally, the interactive tool eases following the ADELFE process.

4 OpenTool for ADELFE

OpenTool is a development tool, written in the OTScript language, which is designed and distributed by TNI-Valiosys, one of the ADELFE partners. On the one hand, OpenTool is a commercialized graphical tool like Rational Rose and supports the UML notation to model applications while assuring that the produced models are valid. More specifically, it focuses on analysis and design of software written in Java. On the other hand, OpenTool enables meta-modeling in order to design specific configurations.

This latter feature has been used to extend OpenTool for taking into account the specificities of adaptive multi-agent systems and thus including them into ADELFE. Therefore, the AUML notation, which allows describing Agent Interaction Protocols using protocol diagrams, has been integrated to OpenTool.

Furthermore, UML has been modified to also express these specificities. Two solutions can be considered to modify the UML notation: using a UML profile or a meta-model extension [5]. Usually this latter solution is chosen when domain concepts are well defined. Nowadays, several different points of view about MAS exist. For example, in ADELFE, we use one specific concept on adaptive multi-agent systems in which agents have to self-organize. But in Tropos or Gaia, agent roles are well-known and the organization can a priori be given. So, UML profiles have been chosen to extend the notation.

Then, in ADELFE, nine stereotypes can be used to modify the semantics of classes and features depending on the specificities of cooperative agents. Including these stereotypes in OpenTool enables the developer to design agents and incorporate them during his modeling activity. Moreover, OpenTool provides several mechanisms to guide designers to correctly implement and use these agents.

Some coherency rules, attached to each stereotype, are defined using the OTScript language and are implemented in OpenTool. OTScript is the action language of OpenTool that can be used to model the behavior of agents; designers have just to express some rules that will be processed by the simulation tool of OpenTool. This enables designers to simulate and check the behavior of the agents that are involved in a system. Moreover, this simulation can be helpful to find if some possible cooperation failures are missing. OpenTool then helps designers to validate the behavior of agents as well as to see if some other points are valid (this corresponds to the activity 17 in Fig. 1):

- Interaction protocols: it is possible to find if some deadlocks can take place within a protocol, or if some protocols are useless or inconsistent... Thus, the behavior of several agents could be judged conform (or not) to the sequence diagrams described in the analysis work definition.
- Methods: their possibly uselessness, their exhaustiveness can be tested.
- The general behavior of agents: to control if this behavior is in accordance with what was expected.

Tests can be realized firstly, by creating the simulation environment in collaboration diagram, and then, by implementing, with OTScript, some methods designers may want to test.

4.1 Modified Class Diagrams

The first modification added to OpenTool concerns the static view of the model: the class diagram. Different ADELFE-specific boxes are defined to take into account the pre-defined stereotypes without decreasing ergonomic aspects of class diagrams because of the large number of available stereotypes. Thus, fields and methods are arranged within boxes in terms of their stereotypes.

Each <<cooperative agent>> stereotyped class must inherit from either the pre-defined CooperativeAgent class or another <<cooperative agent>> stereotyped class. Therefore, each agent class has at least four methods □run, perceive, decide and act □ to simulate its life cycle.

Fig. 2 shows an example of class diagram in which agent classes appear. Since agent classes are object classes, all the operations or associations on object classes are possible on agent classes: composition, aggregation, and heritage. Here, for the ETTO example, the RepresentativeAgent class is composed of several BookingAgent. On the other hand, StudentGroup and Teacher agent classes inherit from RepresentativeAgent class and then inherit its fields and methods.

4.2 Protocol Diagrams

As ADELFE reuses the AUML – Agent Unified Modelling Language [17] – formalism (Agent-UML) to model interactions between agents, OpenTool was enhanced to construct AIP –Agent Interaction Protocol –diagrams. AIP diagrams are an extension to existing UML sequence diagrams that allows different message sending cardinality (AND, OR or XOR). In OpenTool, these diagrams are designed as generic diagrams in the sense that they do not use instances of classes but only roles of classes. This notion of role was added to the OpenTool meta-model.

To respond to ADELFE and AMAS theory specificities, we added some semantics to protocol diagrams. First, a trigger condition is specified to launch the protocol. In the Fig. 3, the BookingAgent with the ExploringTeacherAgent role receives the bookingAgentSeen event as a trigger to begin the negotiation with the OccupyingTeacherAgent.

Moreover, AUML notation remains fuzzy on the XOR and OR branches concerning the decision making of message sending. The choice of attaching an <<aptitude>> stereotyped method to the node (XOR or OR) allows to specify this point. For example, in the Fig. 3, the isRoomFitting method is attached to the first XOR node; i.e., in terms of the result of this method, the ExploringTeacherAgent may either request for information to resume its exploration of the timetable or begin to negotiate in terms of the constraints of the two agents.

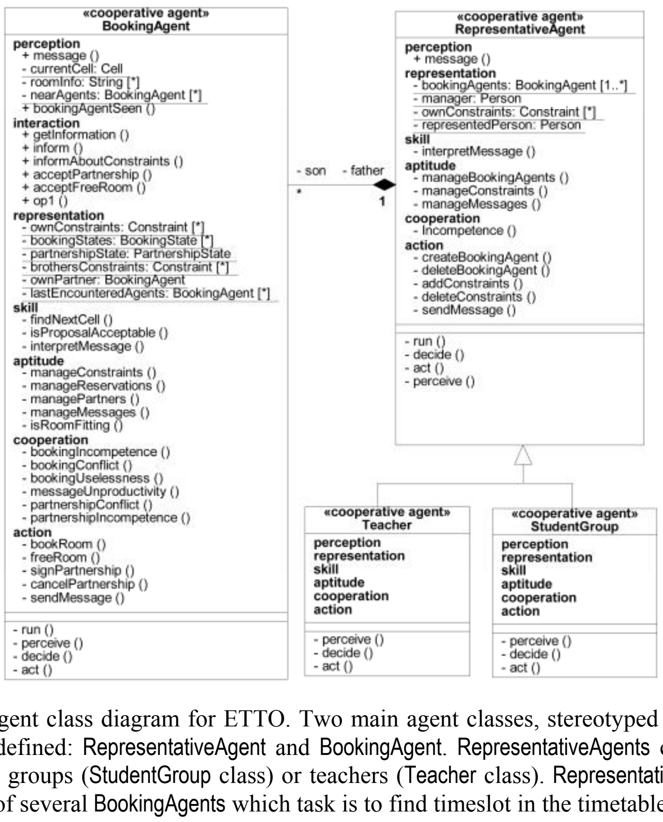


Fig. 2. The agent class diagram for ETTO. Two main agent classes, stereotyped <<cooperative agent>>, are defined: RepresentativeAgent and BookingAgent. RepresentativeAgents can represent either student groups (StudentGroup class) or teachers (Teacher class). RepresentativeAgent class is composed of several BookingAgents which task is to find timeslot in the timetable

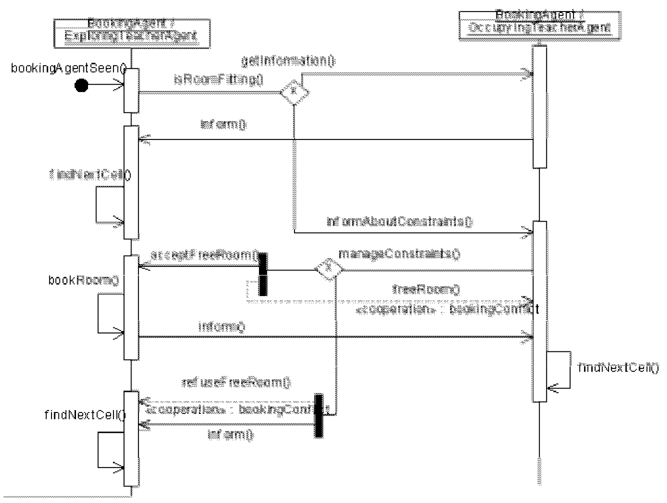
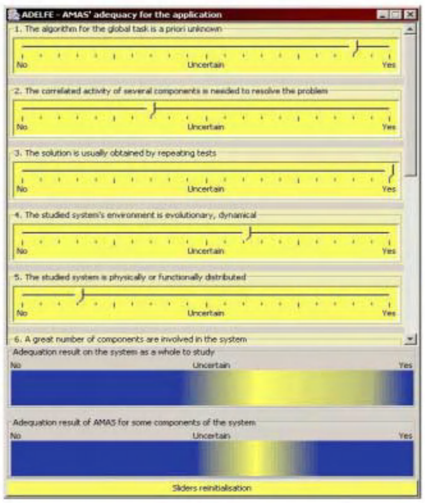


Fig. 3. Here is an example of protocol between two BookingAgents which represent two different teachers. The first agent is exploring the timetable to find valid timeslot and room. The second agent is already occupying a room at a precise timeslot. This protocol diagram explains the negotiation between the two agents. This negotiation may result on the first agent's leaving or on the first agent's booking

Designers can also easily assign protocols to agents in OpenTool. Once a protocol is attributed to an agent for a role, the methods that appear in the protocol for the chosen role are automatically added to the agent class as <<interaction>> stereotyped methods. Finally, to prepare the "Fast Prototyping" activity of the process, designers can automatically generate state-machines – one per role – from the protocol diagrams to simulate them in OpenTool.


ADELFE - AMAS' adequacy for the application

1. The algorithm for the global task is a priori unknown
No Uncertain Yes

2. The correlated activity of several components is needed to resolve the problem
No Uncertain Yes

3. The solution is usually obtained by repeating tests
No Uncertain Yes

4. The studied system's environment is evolutionary, dynamical
No Uncertain Yes

5. The studied system is physically or functionally distributed
No Uncertain Yes

6. A great number of components are involved in the system

Adequation result on the system as a whole to study
No Uncertain Yes

Adequation result of AMAS for some components of the system
No Uncertain Yes

Sliders reinitialisation

Fig. 4. AMAS adequacy graphical tool applied to ETTO

5 AMAS Adequacy Tool

Thanks to activity A11 which has been added in the analysis work definition, designers can know if the application to develop requires to be designed using the AMAS technology. This adequacy is studied both at the global level (the system level) and at the local one (the level of the different parts composing the system, the agents). To study the adequacy of the AMAS theory at the global level, eight criteria have been defined:

1.	Is the global task incompletely specified? Is an algorithm a priori unknown?
2.	If several components are required to solve the global task, do they need to act in a certain order?
3.	Is the solution generally obtained by repetitive tests; are different attempts required before finding a solution?
4.	Can the system environment evolve? Is it dynamic?
5.	Is the system functionally or physically distributed? Are several physically distributed components needed to solve the global task? Or is a conceptual distribution needed?
6.	Does a great number of components needed?
7.	Is the studied system non-linear?
8.	Finally, is the system evolutionary or open? Can new components appear or disappear dynamically?

These questions are asked to designers using a graphical interface which is visualized in Fig. 4. The designer uses a slider to answer a question and to give a rate among twenty possibilities ranging from □yes□to □no□ His answers are then analyzed by the support decision tool to inform him if using an AMAS to implement the system is useful.

The two areas at the bottom of the graphical tool window show the answers of ADELFE regarding the global level and the local one; by clicking on those areas, the user can obtain an interpretation of the results obtained. If the AMAS adequacy tool judges that it is not useful, it can suggest the designer to use another technology; nevertheless, this latter is free to go on using ADELFE.

Considering the ETTO problem, the designer has given to the above criteria the following values: 18, 2, 20, 20, 0, 10, 15 and 15 respectively. The possible answers range from 0 to 20. The answers given to the criteria 1, 3, 5 and 7 are expressing the fact that this problem is a complex and open one for which no well-established algorithm exists. The second and fourth marks are showing that at this point, the designer does not exactly know how to implement a solution; some choices must be made before.

As shown on the Fig. 4, the analysis at the global level expresses that an AMAS is useful to implement ETTO. So, the designer has now to examine the local level to know if some components will need to be recursively viewed as AMAS. If some components need to evolve or adapt themselves, it is rather sure that they will be implemented as AMAS; this explains the three questions at this level:

9.	Does a component only have a limited rationality?
10.	Is a component □big□or not? Is it able to do many actions, to reason a lot? Does it need great abilities to perform its own task?
11.	Can the behavior of a component evolve? Does it need to adapt to the changes of its environment?

Here again, considering the ETTO problem, designers have given 10, 17 and 0 respectively to each one of these three last criteria. The first rate shows that the designer vision is not completely clear about the solution to be adopted. The second one expresses the fact that the different involved components may need to bring the same expertise as a human being. According to the third answer, the system must be adaptive but its components do not need to adapt because, on the one hand, their environment will more or less always be the same, and on the other hand, new types of stakeholders are not going to dynamically appear.

At the local level, because components are coarse-grained, a need for AMAS is also globally shown and designers will need to use the methodology on each component to determine if using an AMAS is required to implement this component (Fig. 1). This adequacy tool can be used in an autonomous manner but is also linked to the interactive tool that is now described.

6 Interactive Tool

ADELFE also provides an interactive tool that helps designers when following the process established in the methodology.

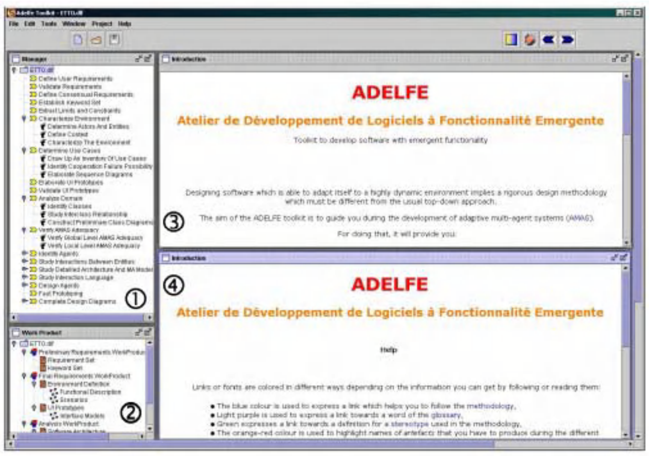
In classical object-oriented or agent-oriented methodologies, this kind of tool does not really exist. Even if some tools linked with agent-oriented methodologies exist (e.g. AgentTool 5 for MaSE, PTK for PASSI or the INGENIAS tool [15]), they are not really a guide and a verification tool for designers following a methodology process.

Generally, some guides like books or html texts are given (e.g., a guide to follow the RUP is available on the web site of Rational Software, http://www.rational.com/products/rup/) but they are not really interactive tools able to follow a project through the different activities of the process. The ADELFE interactive tool is linked both with the AMAS adequacy tool and with OpenTool. It can communicate with OpenTool in order to access to different diagrams as process progresses. For these two reasons, it can be considered as a real guide that supports the notation adopted by the process and verifies the project consistency.

Each activity or step of the process is described by this tool and exemplified by applying it to the ETTO problem. Within the textual description or the example, some AMAS theory specific terms can be attached to a glossary in order to be explained. That is why the interactive tool is composed of several interfaces which will be detailed in the following paragraphs:

- A □Manager□ interface indicates, for the different opened projects, the different activities and steps designers have to follow when applying the methodology,

- A “Work Product” interface lists the work products that have been produced or that have to be produced yet regarding the progress when applying the methodology,
- A “Description” interface explains the different stages (activities or steps) designers must follow to apply the methodology process,
- An “Example” interface shows how the current stage has been applied to ETTO,
- An optional “Synthesis” interface shows a global view and an abstract of the already made activities,
- An optional “Glossary” interface explains the terms used in the methodology and defines the stereotypes that have been added to UML.


Adelfe Toolkit - ETTO.dlf
File Edit Tools Window Project Help
Manager
ETTO.dlf
Define User Requirements
Validate Requirements
Define Consensual Requirements
Establish Keyword Set
Extract Limits and Constraints
Characterize Environment
Determine Actors And Entities
Define Context
Characterize The Environment
Determine Use Cases
Draw Up An Inventory Of Use Cases
Identify Cooperation Failure Possibility
Elaborate Sequence Diagrams
Elaborate UI Prototypes
Validate UI Prototypes
Analyze Domain
Identify Classes
Study Interclass Relationship
Construct Preliminary Class Diagrams
Verify AMAS Adequacy
Verify Global Level AMAS Adequacy
Verify Local Level AMAS Adequacy
Identify Agents
Study Interactions Between Entities
Study Detailed Architecture And MA Model
Study Interaction Language
Design Agents
Fast Prototyping
Complete Design Diagrams
①
Work Product
ETTO.dlf
Preliminary Requirements WorkProduct
Requirement Set
Keyword Set
Final Requirements WorkProduct
Environment Definition
Functional Description
Scenarios
UI Prototypes
Interface Models
Analysis WorkProduct
Software Architecture
②
Introduction
③
④
Introduction

ADELFE

Atelier de Développement de Logiciels à Fonctionnalité Emergente

Toolkit to develop software with emergent functionality

Designing software which is able to adapt itself to a highly dynamic environment implies a rigorous design methodology which must be different from the usual top-down approach.

The aim of the ADELFE toolkit is to guide you during the development of adaptive multi-agent systems (AMAS).

For doing that, it will provide you:

ADELFE

Atelier de Développement de Logiciels à Fonctionnalité Emergente

Help

Links or fonts are colored in different ways depending on the information you can get by following or reading them:

- The blue colour is used to express a link which helps you to follow the methodology.
- Light purple is used to express a link towards a word of the glossary.
- Green expresses a link towards a definition for a stereotype used in the methodology.
- The orange-red colour is used to highlight names of artefacts that you have to produce during the different

Fig. 5. The ADELFE interactive tool mainly consists of four modules: the Manager interface (1), the Workproduct interface (2), the Description interface (3) and the Example interface (4)

A global view of the tool, excluding the optional windows, is given in the Fig. 5.

Manager Interface. The interface called “Manager” displays, using a directory structure, the different already processed activities and steps for the end-user's opened projects. To expand the directory and to display the list of activities and steps, a user has to select one existing project (or to create one if there is none). He can click on a name of an activity or step to work on it. He can also backtrack in the process of the methodology to manage lifecycle as he wants. For example, on the Fig. 5, module ①, and for the project ETTO, ADELFE displays all activities and details the steps of the “Characterize Environment”, “Determine Use Cases”, “Analyze Domain” and “Design Agents” activities.

Work Product Interface. The interface “WorkProduct” displays the list of the documents about the current project, as shown in Fig. 5, module ②. In this list, the name of a document is colored depending on the current state of this document:

- A document that cannot be realized during a step for the current project is marked in grey. For example, the project has not made enough progress to realize it.

- A document that has already been produced is colored in green.
- And red is used to mark a document that could be realized at the previous or at the current step but is not realized yet.

This list is dynamically updated whenever new documents are produced.

Description Interface. Being a guide to realize a given current stage (activity or step), this window, module ③ in Fig. 5, displays several types of information such as:

- The stage name such as mentioned in the ADELFE process,
- A brief description about the objectives of this stage, how designers must proceed to complete this stage and what are the work products to generate during it,
- Links towards the “Glossary” window if some terms need to be characterized,
- Links towards other stages, because backtrackings during the process is possible,
- Links to different tools (such as OpenTool, the AMAS adequacy tool, a text editor…) to enable designers to access them. These links are mentioned within the textual description whenever designers need to access a piece of software to achieve the current step. Moreover, if specific entities or diagrams in OpenTool must be created, updated or completed, the interactive tool is able to communicate with OpenTool in order to make the designer access to the right diagrams. For instance, when designers need to describe use cases within the activity 7 (“Determine Use Cases”), an OpenTool icon may be clicked to directly access the window displaying use cases in the OpenTool software.

Example Interface. Thanks to the “Example” window, module ④ in Fig. 5, designers can obtain an additional help because this window displays what has been done to apply the ADELFE methodology regarding the current stage. As mentioned before, this window mainly refers to the ETTO application but other specific applications that apply the AMAS theory could also be used to give some complementary information. For instance, some specific concepts in the AMAS theory cannot be easily apprehended and different points of view can be used to explain them.

Synthesis and Glossary Interfaces. Two optional windows that are displayed in a separate way also exist in the interactive tool. The first one, named “Synthesis” gives a global view of the current project when applying the methodology process. It presents a compilation of the different work products produced so far: textual descriptions, UML diagrams created with OpenTool, GUI specifications… The content of this window represents a help to prepare the final project documentation. The second one, named “Glossary”, displays and explains a list of terms that are related to the methodology process or to the AMAS theory. Stereotypes added to UML and OpenTool are described as well as the rules that must be applied when using them.

7 Conclusion

For several years now, we had worked on a theory to build self-organizing systems. This paper has very briefly expounded the AMAS theory that has been successfully applied to several open and complex problems for which no solution was known in advance [8]. The specificity of the theory resides in the fact that the global function of the system is not coded within the agents but emerges from the collective behavior of the different agents composing it. Each agent can locally rearrange its interactions with others depending on the individual task it has to solve and cooperation failures it may encounter and process. This capacity of self-organization by cooperation at the lowest level enables to change the global function without coding this modification at the upper level of the system.

A further step, in our research, has been to study how a self-organizing system could be engineered to ease the work of a designer and promote this technology beyond academia frontiers. This work has led to ADELFE, a toolkit to develop self-organizing applications, which can guide and help an engineer during such a task. ADELFE differs from other agent-based methodologies because it is specific to self-organizing applications. The process is based on the RUP enriched with some specific steps dedicated to agents and to adaptive multi-agent systems. To apply a methodology process, some textual description in books or html forms can be found but, in a generally manner, the process is not interactive. The originality of the ADELFE tool associated to the process resides in the interaction with designers. Moreover, it provides a particular important tool, OpenTool, which supports the UML and AUML notations, and enables the designer to draw his graphical UML diagrams. It also checks the coherence and the completeness of the specification and design. A first version of the ADELFE is now operational (and can be downloaded from http://www.irit.fr/ADELFE); it was used for designing the ETTO application. In the near future, this prototype will be used to design several other complex systems such as a mechanical design system or a flood forecast system.

Acknowledgements

We would like to thank the support of the French Ministry of Economy, Finances and Industry as well as our partners: TNI-Valiosys Ltd., for their customization of OpenTool, ARTAL technologies Ltd., for their help on the ETTO application design, and the IRIT software engineering team, for their help and work concerning UML and SPEM.

References

[1] Bernon C., Gleizes M-P., Peyruqueou S., and Picard G., ADELFE: a Methodology for Adaptive Multi-Agent Systems Engineering, *Third International Workshop "Engineering Societies in the Agents World" (ESAW-2002)*, 16-17 September 2002, Madrid.

[2] Burke E.K., and Petrovic S., Recent Research Directions in Automated Timetabling, *European Journal of Operational Research - EJOR*, 140/2, 2002, 266-280.

[3] Castro J., Kolp M., and Mylopoulos J., A Requirements-driven Development Methodology, In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, Stafford, UK □June, 2001.

[4] Cossentino M., Different Perspectives in Designing Multi-Agent System, *AgeS'02 (Agent Technology and Software Engineering) Workshop at NodE'02*, Erfurt, Germany, Oct 2002.

[5] Desfray P., UML Profiles Versus Metamodel Extensions : An Ongoing Debate, *OMG's UML Workshops: UML in the .com Enterprise: Modeling CORBA, Components, XML/XMI and Metadata Workshop*, November 2000.

[6] DeLoach S.A., and Wood M., Developing Multiagent Systems with agentTool, in Intelligent Agents VII. AgentTheories Architectures and Languages, *7th International Workshop (ATAL 2000, Boston, MA, USA, July 7-9, 2000)*, C. Castelfranchi, Y. Lesperance (Eds.). LNCS Vol. 1986, Springer Verlag, Berlin, 2001.

[7] Eurescom, Project P907-GI - *MESSAGE: Methodology for Engineering Systems of Software Agents, Deliverable 1 - Initial Methodology*, http://www.eurescom.de/~pub-deliverables/P900-series/P907/D1/P907D1.

[8] Gleizes M-P., Glize P., R□gis C., and Sontheimer T., Real-time Simulation for Flood Forecast : an Adaptive Multi-Agent System STAFF, in AISB□03, Aberystwyth, April 2003.

[9] Gleizes M-P., Glize P. and Link-Pezet J., An Adaptive Multi-Agent Tool For Electronic Commerce, In *The workshop on Knowledge Media Networking IEEE Ninth International Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprises (WET ICE 2000)* 14-16 June 2000 Gaithersburg, Maryland.

[10] Gleizes M-P., George J-P., and Glize P., A Theory of Complex Adaptive Systems Based on Co-operative Self-Organisation: Demonstration in Electronic Commerce, *Self-Organisation in Multi-Agent Systems (SOMAS)* July 27-28 2000, Milton Keynes UK, A Workshop organised by the Emergent Computing Network.

[11] Gleizes M-P., Camps V., and Glize P., A Theory of Emergent Computation Based on Cooperative Self-Oganization for Adaptive Artificial Systems, *Fourth European Congress of Systems Science*, Valencia, 1999.

[12] Jacobson I., Booch G. and Rumbaugh J., *The Unified Software Development Process*, Addison-Wesley, 1999.

[13] Heylighen F., The Science of Self-organization and Adaptivity, In The *Encyclopedia of Life Support Systems*, (EOLSS Publishers Co. Ltd), 2001.

[14] Kinny D., Georgeff M., and Rao A., A Methodology and Modelling Technique for Systems of BDI Agents, In Van Der Velde, W., Perram, J. (eds.): *Agents Breaking Away: Proc. of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World* (LNAI Vol. 1038), 1996.

[15] Gomez Sanz J., and Fuentes R., Agent Oriented System Engineering with INGENIAS, in *Fourth Iberoamerican Workshop on Multi-Agent Systems, Iberagents* 2002.

[16] Link-Pezet J., Gleizes M-P., and Glize P., FORSIC: a Self-Organizing Training System - *International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000)* December 11-13, 2000, Wollongong, Australia.

[17] Odell J., Parunak H.V., and Bauer B., Extending UML for Agents, In *Proceedings of the Agent Oriented Information Systems (AOIS) Workshop at the 17th National Conference on Artificial Intelligence (AAAI)*, 2000.

[18] OMG, *Software Process Engineering Metamodel Specification*, http://cgi.omg.org/docs/formal/02-11-14.pdf.

[19] Picard G., and Gleizes M-P., An Agent Architecture to Design Self-Organizing Collectives: Principles and Application, In *Proceedings of the AISB'02 Convention Symposium on Adaptive Agents and Multi-Agent Systems*, University of London, 3rd-5th April 2002.

[20] Piquemal-Baluard C., Camps V., Gleizes M-P., and Glize P., Cooperative Agents to Improve Adaptivity of Multi-Agent Systems, *Intelligent Agents Workshop of the British Computer Society*, Edited by N.S.Taylor and J.L.Nealon, Oxford, November 1995.

[21] Russel S., and Norvig P., *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.

[22] Socha K., Sampels M., and Manfrin M., Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art, *Proceedings of 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization (EvoCOP'2003)*, Essex, UK, April 14-16, 2003.

[23] Topin X., Fourcassie V., Gleizes M-P., Theraulaz G., Regis C., and Glize P., Theories and Experiments on Emergent Behaviour: From Natural to Artificial Systems and Back, *Proceedings on European Conference on Cognitive Science*, Siena, 1999.

[24] Wooldridge M., Jennings N.R., and Kinny d., A Methodology for Agent-Oriented Analysis and Design, In *Proceedings of the 3rd International Conference on Autonomous Agents (Agents 99)*, pp 69-76, Seattle, WA, May 1999.