Lecture Notes in Computer Science 3002

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

David L. Hicks (Ed.)

Metainformatics

International Symposium, MIS 2003
Graz, Austria, September 17-20, 2003
Revised Papers

Springer

Volume Editor

David L. Hicks
Aalborg University Esbjerg
Department of Software and Media Technology
Niels Bohrs Vej 8, 6700 Esbjerg, Denmark
E-mail: hicks@cs.aue.auc.dk

Library of Congress Control Number: 2004104702

CR Subject Classification (1998): H.4, H.5.1, D.2, H.5.4, D.1, I.2, K.4, I.7

ISSN 0302-9743
ISBN 3-540-22010-0 Springer-Verlag Berlin Heidelberg New York

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

springeronline.com

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11008378 06/3142 5 4 3 2 1 0

Preface

This volume contains the final proceedings of the MetaInformatics Symposium 2003 (MIS 2003). The event was held September 17–20 on the campus of the Graz University of Technology in Graz, Austria.

As with previous events in the MIS series, MIS 2003 brought together researchers and practitioners from a wide variety of fields to discuss a broad range of topics and ideas related to the field of computer science. The contributions that were accepted to and presented at the symposium are of a wide variety. They range from theoretical considerations of important metainformatics-related questions and issues to practical descriptions of approaches and systems that offer assistance in their resolution. I hope you will find the papers contained in this volume as interesting as the other members of the program committee and I have.

These proceedings would not have been possible without the help and assistance of many people. In particular I would like to acknowledge the assistance of Springer-Verlag in Heidelberg, Germany, especially Anna Kramer, the computer science editor, and Alfred Hofmann, the executive editor for the LNCS series.

February 2004 David L. Hicks

Organization

Organizing Committee

Uffe K. Wiil (Aalborg University Esbjerg, Denmark)
David L. Hicks (Aalborg University Esbjerg, Denmark)
Peter J. Nürnberg (Technical University Graz, Austria)

Conference Secretary

Mathias Lux (Know-Center, Austria)

Program Committee

Chair	David L. Hicks (Aalborg University Esbjerg, Denmark)
Members	Darren Dalcher (Middlesex University, London, UK)
	David Millard (University of Southampton, UK)
	Peter J. Nürnberg (Technical University Graz, Austria)
	Siegfried Reich (Salzburg Research, Austria)
	Jessica Rubart (FhG-IPSI, Darmstadt, Germany)
	Klaus Tochtermann (Know-Center, Graz, Austria)
	Manolis Tzagarakis (University of Patras, Greece)
	Weigang Wang (FhG-IPSI, Darmstadt, Germany)
	Jim Whitehead (University of California, Santa Cruz, USA)
	Uffe K. Wiil (Aalborg University Esbjerg, Denmark)

Sponsoring Institutions

Technische Universität Graz, Austria
Styrian Competence Center for Knowledge Management (Know-Center), Austria
Aalborg University Esbjerg, Denmark

Table of Contents

A Grand Unified Theory for Structural Computing

Peter J. Nürnberg*, Uffe K. Wiil, and David L. Hicks

Department of Computer Science, Aalborg University Esbjerg
Niels Bohrs Vej 8, DK-6700 Esbjerg, Denmark
{pnuern,ukwiil,hicks}@cs.aue.auc.dk

1 Introduction

Structural computing, in one sense, seeks to unify the notions of data and structure under a synthesized abstraction, by which data and structure become *views* to be applied as the need or desire arises. Indeed, one way of looking at structural computing is that the notions of data and structure are contextual, not essential. Any entity may be data to one person (application, agent, whatever) at one moment, and structure to another. Data and structure are matters of interpretation, not essence. What exactly this has bought us is discussed at length elsewhere [7,10,11].

However, despite the presence of the term "computing," structural computing research has left behavior (computation) out in the cold. Many agree that "behavior is important" [2,13], but exactly how this abstraction should be integrated into the broader picture is unclear.

In this paper, we embark on a thought experiment: how can the notion of behavior be integrated into the data/structure synthesis, to derive a "grand unified abstraction?" Is behavior, like data and structure, non-essential – merely the product of a view to be applied as the need arises? If so, how can unifying data, structure, and behavior potentially benefit us?

It must be said up front that this thought experiment is in its early stages. Nonetheless, we present this work in the belief that it is sufficiently mature to benefit from active discussion within this field. Furthermore, we feel that the applications to metainformatics are clear: if this unification can succeed and is beneficial, it stands to impact a broad variety of fields, and potentially draw in researchers from hereto "foreign" fields such as computational theory.

The remainder of the paper is organized as follows: Section 2 presents a review of the data/structure unification undertaken thus far within structural computing. Section 3 describes one extremely simple (but Turing complete) behavior model and a prototypic implementation of this model in Java. Section 4 considers the synthesis of behavior and the data/structure abstraction. Section 5 considers some connections between the work presented here and other fields.

* The first author kindly acknowledges the Styrian Competence Center for Knowledge Management Research for partially supporting the work presented here. Please see http://www.know-center.at/ for more details.

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 1–16, 2004.

Section 6 concludes the paper with discussions of future directions for this research.

2 Data/Structure Synthesis

In [8], it is claimed that hypermedia is about structure, but that most hypermedia systems treat structure as a derived abstraction. Data underlies most hypermedia systems. Structure is something built out of data. In this view of the world (prevalent not only in hypermedia but in almost every field) data is atomic. Some instances of data abstractions might be interpreted by some systems in certain ways. This implies that, at the backend, what is needed are generalized data management systems. Infrastructure that manages data is sufficient for hypermedia (and, indeed, any) "higher-level" application.

Structural computing disagrees with this assertion. Hypermedia is not just an application, like spreadsheet building or document management. It is an expression of an epistemological point of view – that *all* knowledge work is structuring work. Since hypermedia is about structure, it represents not as much an application as a radical rethink of the way systems (intended for human users, anyway) should be designed. Such systems need to account for the omnipresence of structure. Furthermore, this structure is arbitrary in nature. Associations built by humans among concepts are arbitrary (not regular in an obviously formalizable and computable way). Systems should support this non-formal, non-computable, arbitrary structure at the backend. Doing this is an attempt to make the technology of the computer fit the way people think, instead of forcing people to think as the computer is designed.

This has certain fairly radical implications for the way in which structure and data are related to one another. In the traditional "data-centric" view of systems construction, structure is an abstraction generated at the middleware layer at runtime. Data is an essential characteristic of all entities managed by systems. In the structural computing view, both data and structure are runtime views. As such, any entity may stand in relation to other entities, or even represent relations among other entities, at any time. Indeed, system infrastructure must assume that any entity is at all times (at least potentially) both *structured* and *structuring.*

What does this mean? Examples of interpretations of the structured and structuring roles (or in short, the structural nature) of entities are numerous. A short list here can serve to refresh the reader's memory: versioning; notification; access control; and, concurrency control – all have been shown to be non-trivially more complicated when managed entities are handled structurally.

What is the "atomic" abstraction in structural computing systems? Traditionally, different research groups have named this abstraction differently, but most names imply a highly structural view: abstract structural element [16] or structural unit [18] are two fine examples. This belies the fact, however, that the atomic abstraction is more correctly seen as a *synthesis* of data and structure. It is not as if structural interpretations of, say, versioning *replace* data interpretations, but rather that they augment them. In this view, more appropriate names for the basic atomic abstraction might be the "dataic/structural atom" or "dataic/structural unit" – with the obvious caveat that this is, to say the least, awkward. Nonetheless, the point is clear: structural computing advocates a data/structure synthesis as the system atomic abstraction to be managed at the infrastructure layer of systems.

There are benefits that are immediately derived from such a viewpoint. Indeed, much structural computing research focuses on these benefits rather than any theoretical reasoning for such system constructions in the first place [1,17]. Some researchers have gone so far as to claim that finding and articulating such benefits should be the main priority of the field [3]. Such claimed benefits usually fall into one of three categories: omnipresence of structural support (at the middleware layer) – the so-called "convenience" arguments; efficiency of structural operation implementation – the aptly named "efficiency" arguments; or, the guaranteed basic structural interpretation of entities – the so-called "interoperability" arguments.

Convenience arguments, such as those presented in [1], claim that by providing structural abstractions at the infrastructure layer, middleware services, such as software management systems, that export structural abstractions (and operations) are more easily (and consistently) implemented.

Efficiency arguments, such as those presented in [9], claim that by sinking the implementation of structural operations into the infrastructure, certain operations may execute more efficiently, even operation that are seemingly (or traditionally viewed as) non-structural, such as prefetching and caching.

Interoperability arguments, such as those presented in [6], claim that by guaranteeing some basic structural interpretation to be present at all times in the infrastructure, recasting of structural entities specialized for one application domain (e.g., navigational hypermedia) into those for another (e.g., spatial hypermedia) may be made easier.

Behind all of these arguments lies the basic claim that the data/structure synthesis is a better atomic abstraction that a solely data-oriented abstraction. Yet despite the nearly universal claim of the fundamental importance of behavior, the nearly exclusive focus on the data/structure synthesis has left behavior as a "residual," an unanalyzed concept of second-class importance.

In this paper, then, we make the following claim: analogous to the rather curious second-class, derived status of structure in hypermedia (a field ostensibly about structure), behavior occupies a second-class, derived status in structural computing (a field arguably as much about behavior as structure or data). Here, we look at some first steps toward remedying this state of affairs.

3 OISC – A Simple Turing-Complete Behavior Model

In this section, we present a very simple behavior model, which has the desirable characteristic of being Turing-complete. It is not yet clear whether this is a sufficient behavior model in practice (almost certainly this is not the case), but it clearly is sufficient in theory. Any theoretically complete model can serve as a basis for more practical and useful models through transformation of these more complex models into their simple counterpart.

3.1 Defining subleq

OISC stands for "one instruction set computer" – analogous to the well-known CISC and RISC abbreviations [19]. The OISC instruction "set" indeed consists of only one instruction, but is nonetheless Turing-complete[1]. This instruction takes three parameters and is concisely described as "subtract and branch on result less than or equal to zero," abbreviated `subleq a b c`. (Hereafter, both the instruction set and any language based upon it are referred to as "subleq.") In "C" style pseudocode, this instruction does the following:

```
*b-= *a;
if (*b <= 0) goto c;
```

Oddly enough, subleq "programs" consist of parameters only, since no instruction operation codes are needed. "Well-formed" subleq programs, then, consist of triples of parameters, in which the first two parameters name locations to be manipulated algebraically, and the third names a potential jump location. We don't define "well-formed" here, but intuitively, we mean programs for which no parameter operates both as a algebraic location and as a jump location (though of course nothing prevents this from potentially being the case in arbitrary subleq programs.)

Like most minimal Turing-complete languages, subleq isn't very useful by itself. For example, there is no "halt" instruction, meaning the "program counter" (tape head) must be run off the end of the tape to halt the machine. There are no I/O instructions either, meaning the "memory" (tape) on which a program runs must be initialized (somehow). These, theoretically speaking, are details, however.

For simplicity and brevity, the `subleq a b` is to be interpreted as implying a "c" value that corresponds to whatever value to the PC would be if the jump condition failed. That is, the shorthand `subleq a b` implies that no jump is executed, regardless of the value of "b."

3.2 A Prototypic Implementation of subleq

We have built a prototypic implementation of a subleq virtual machine and a simple operating system and assembler for use with this virtual machine to experiment with subleq. This implementation is available under the GNU GPL from the Department of Computer Science at Aalborg University Esbjerg. Please see <`http://cs.aue.auc.dk/~pnuern/subleq/`> for more details. A description of this implementation is presented here.

[1] The proof of this is left as an exercise for the reader!

Virtual Machine. There is little to say about the virtual machine (VM) – it has only one instruction! When a VM instance is created, it has a fixed amount of memory. This memory is conceptually a one-dimensional array of integers, initially set to all zeroes. Program images (integer arrays) may be loaded into memory at arbitrary locations. The VM is able to execute the subleq instruction at any arbitrary given location in memory and supports "core dumping" for any range of memory. The VM will halt if asked to execute subleq at any "out of bounds" memory location.

Operating System. The operating system (OS) is substantially more complicated than the VM. Strictly speaking, the OS is an operating system "simulation," since it is neither implemented in subleq, nor runs on the VM. Nonetheless, it performs many of the functions of a very simple OS, and "appears" as an OS for the subleq VM to subleq programmers.

The basic abstraction of the OS is that of *process*, which consists of zero (sic) or more instances of the *thread* abstraction. A process is defined by: a range in memory; a set of threads; a name; and, a symbol table. A thread is defined by a program counter. Note that these abstractions are somewhat "poorer" than traditional notions of these same terms. Processes are not, for example, divided into text, data, stack, and heap sections. Processes may freely(!) modify their own "text" section memory, have fixed sizes (no dynamic allocation), and are responsible for all "free space" management themselves. Threads do not have register sets (there are no registers in the subleq VM), call stacks (these, also, do not exist in the VM) or any thread specific data. Processes with zero threads are also non-traditional – such "processes" are essentially global data.

The OS (intentionally) enforces no protection for process memory (with exceptions, see below). Instructions may freely reference memory locations outside the process (memory range) in which the instruction itself is executed.

There are two "special" processes – the *I/O* process and the *system* process. In the current implementation, both are 48 integers large, and are automatically loaded into locations 0-47 and 48-95, respectively. Program counter values inside these processes' memory space cause the VM to halt. (Technically, the OS translates all instructions that attempt to set the PC to a location inside these processes to "equivalent" instructions that set the PC to −1, which will cause the VM to halt.) These special processes are explained in more detail below.

I/O Process. Read and write operations to I/O locations behave specially. The 48 locations of the I/O block are to be interpreted as references to (up to) 48 I/O streams. The first 3 locations in the I/O block are hardwired to standard in, standard out, and standard error, respectively. References to I/O locations in the "a" parameter location result in a "read" from the appropriate I/O stream – the value read substitutes for the the value stored at "a". References to I/O block locations in the "b" parameter location result in a "write" of the "a" value to the appropriate I/O stream. More explicitly, consider the following subleq instructions and their "C" pseudocode counterparts:

```
subleq 0 b
   *b-= read(0);

subleq a 1
   write(1, *a);
```

A few notes are necessary. These examples use the "implicit c parameter" convention. This is not strictly necessary. The semantics of both operations are not necessarily apparent intuitively, but they do effect a primitive I/O semantic that is effective. Also note that IPC requires no I/O, since in principle, all processes within a distributed I/O instance may perform "virtual" I/O with one another using a shared memory model. More on this below.

Currently, there is no way to open or close new streams – thus, only the 3 hardwired stream are available for I/O. Also, there are no threads in the I/O process. "Write" references to read-only streams (e.g., stdin) or "read" references to write-only streams (stdout or stderr) result in an OS halt (`InappropriateStreamAccessException`). Currently, any reference to I/O process locations other than to the hardwired 3 special locations results in an OS halt (`IllegalIOAccessException`).

System Process. The system process, like the I/O process, has no threads. Instead, it consists of a few memory locations that the OS interprets specially when referenced as the "a" or "b" parameters of an instruction. Specifically, there are three memory locations, at relative offset 0, 1, and 2, which are specially interpreted: relative 0 is guaranteed to hold the value "0" when read; relative 1, "+1"; relative 2, "−1". All three of these locations are guaranteed to "ignore" writes. References to any other location in this address space result in an OS halt (`IllegalSystemAccessException`).

Assembler.

Assembler Directives. The assembler accepts three "directives" that have no translation into subleq instructions. Instead, they modify the behavior of the assembler during assembly and/or load time. These directives are:

```
:prog <progname>
:var <varname>
:loc <locname>
:thread <locname>
```

The `:prog` directive provides the name of the process that executes the assembled image that corresponds to the assembler program. It must appear exactly once in a valid program. The `:var` directive instructs the assembler to allocate a location in the assembled image, and to translate all symbolic references to the declared varname into the relative address of the allocated location. Every varname must be unique. The `:loc` directive instructs the assembler to translate all jump requests that refer to the locname symbol into relative address of the instruction following the directive. Every locname must be unique. All names must take the form of Java identifiers. All varnames and locnames exist in the same name space. (That is, there may not be identically named variables and locations.) The `:thread` directive instructs the operating system to create a thread with the program counter initially set at the given location.

Details on Symbols. A few notes on variables and locations. Normally, references to variables or locations are to places within the process (memory range) containing the referring instruction. However, this need not be the case. As mentioned above, processes may access one another's memory. This is accomplished as follows.

Each process contains a symbol table, which binds symbols to relative offsets. (This table is trivially constructed during assembly.) All symbol references are resolved at *loadtime* by the OS, as explained below. Symbols are generally of the form *processname.symbolname.* The *processname* qualifier may be omitted for references to symbols declared within the same program as the referring instruction.

There are 6 "predefined" symbols. These symbols, and their resolutions, are:

`io.stdin`	0
`io.stdout`	1
`io.stderr`	2
`system.zero`	48
`system.posone`	49
`system.negone`	50

Note that it is only possible to name the resolutions of these symbols a priori since it is (currently) fixed that the I/O and system processes are loaded at locations 0 and 48, respectively. This may change in the future, so these symbols, just as all others, are in practice resolved at load time.

Assembler Instructions. The assembler implements a slightly more complex instruction set than the VM. Specifically, the *additional* instructions available to assembler programmers (and their equivalent subleq implementations) are as follows:

Algebraic instructions

```
zero <varname>
   subleq <varname> <varname>
inc <varname>
   subleq system.negone <varname>
dec <varname>
   subleq system.posone <varname>
```

Program flow instructions

```
halt
   subleq system.zero system.zero -1
jump <locname>
   subleq system.zero system.zero <locname>
incloop <varname> <locname>
   subleq system.negone <varname> <locname>
decloop <varname> <locname>
   subleq system.posone <varname> <locname>
```

I/O instructions

```
read <streamname> <varname>
   subleq io.<streamname> <varname>
write <streamname> <varname>
   subleq <varname> io.<streamname>
```

Miscellaneous instructions

```
nop
   subleq system.zero system.zero
```

Assemble-Time. When an assembler program is assembled, all instructions are transformed into their subleq equivalents, all `:var` and `:loc` directives are used to generate a symbol table, appropriate free space is allocated (and initialized to zero) for all variables, the process name is stored, and a thread descriptor block (containing start locations for each declared thread) is built. The resultant object code can then be loaded at a later time.

Load-Time. When an image is loaded, the OS must first resolve all symbols in the object code. For each symbol of the form *processname.symbolname*, the OS tries to find a process with the given name (failure to find the appropriate process results in termination of the load with an `IllegalProcessReference-Exception`), then queries its symbol table for the appropriate symbol (failure to find the symbol results in load termination with an `IllegalSymbolReference-Exception`), and then adds the resultant relative offset to the process image offset to generate an absolute address. Once all symbols are resolved and all instructions translated into absolute address form, a thread is started at the appropriate location for each entry in the thread descriptor block.

Runtime. Even with all addresses in absolute form, the OS is still active in process management at runtime. Specifically, when processes are loaded, they are normally flagged as "user" processes, in which case, instructions are executed directly by the VM. However, both the I/O and the system processes are loaded into the VM in "supervisor" mode, meaning all reads, writes, and jumps from/to these locations are passed to the OS to handle instead of the VM. Blocks loaded in supervisor mode must also pass an instance implementing the `Operating-System` interface, which primarily consists of the following methods:

```
int read(int loc);
int write(int loc, int value);
int jump(int loc);
```

When reads to locations within supervisor processes are made, the VM executes the `read` callback to the registered Operating System implementation, which then returns the value that should be "read" from the given location. The `write` callback should return the value to be stored in a location. The `jump` callback should return an appropriate PC value, if an instruction wants to jump to the given location. As examples, a "passthrough" implementation of this interface (which acts just as the VM would) could look as follow:

```
public int read(int loc)
{ return _memory[loc]; }

public int write(int loc, int value)
{ _memory[loc] = value; return _memory[loc]; }

public int jump(int loc)
{ return loc; }
```

As a further example, the current implementation of operating system read method (in pseudocode) that handles the I/O process looks as follows:

```
public int read(int loc)
{
   switch(loc) {
     case 0:
      return read(0);
     case 1:
     case 2:
      throw new InappropriateStreamAccessException();
   }
   throw new IllegalIOAccessException();
}
```

An Example Assembler Program. The following program reads two positive numbers from standard in, multiples them, and writes the result to standard out:

```
# program to calculate prod = num1 * num2
# note: num1 and num2 must be non-negative
:prog mymult
:var num1
:var num2
:var prod
:var negnum2
:thread start
```

```
:loc start
# read the input
read io.stdin num1
read io.stdin num2
# test the input
inc num1
subleq system.zero num1 badnum1
dec num1
inc num2
subleq system.zero num2 badnum2
dec num2
# build a "negative copy" of num1.  There is no add in subleq,
# so we effect addition by subtracting a negative copy.
subleq num2 negnum2
# we will add num2 to prod, num1 times
# first, add one to num1 so the loop condition works
inc num1
# loop: decrement num1 and branch on zero
:loc beginloop
decloop num1 endloop
   subleq negnum2 prod
   jump beginloop
# we're done: output the result and halt the thread
:loc endloop
write io.stdout prod
halt
:loc badnum1
write io.stderr num1
halt
:loc badnum2
write io.stderr num2
halt
```

4 Data/Structure/Behavior Synthesis

We initially set out to synthesize the abstractions of data, structure, and behavior into one grand unified atomic abstraction. We now present one such possible synthesis, and describe its ramifications.

4.1 Defining DSB I: A First Grand Unified Synthesis

The initial definition of the data/structure synthesis started out with the question: What would it mean for any entity to be potentially *structured* and *structuring* at any time? This led to the notion of "structural units" or other such abstractions that included innate structuring mechanisms. In this model, "data" is simply unexplicated structure. Better stated, when an abstraction is viewed as data, its structuring properties are ignored.

The analogous starting point for a data/structure/behavior synthesis might be the question: In addition to the above, what would it mean for an entity to be potentially *computed* and *computing* at any time? This might lead to the notion of "DSB" (data/strucure/behavior) units. In this model, the data view ignores structure and computation properties; the structural view ignores data and computation properties, and the behavior view ignores the data and structure properties.

There are many unresolved questions. What are "data," "structure," and "behavior" properties? Are they properly "ignored" in various views? There are potentially many ways to flesh out this definition. We propose one here, which we call DSB I.

Before we begin, it should be noted that the DSB I synthesis, like structural computing data/structure syntheses, is an essentially infrastructure abstraction. Its interpretations at the middleware or application layers of systems are not relevant (because they are arbitrary). Also, we only speak of essential characteristics of DSB I below. We will ignore many details, such as how instances of these entities are efficiently authored, stored, retrieved, etc.

Data View and Properties. The notion of data view has often gone unanalyzed (at least explicitly) in most structural computing work. We will propose a very simple model of data that seems to fit the implicit notion in most current structural computing models. The data view of an entity highlights (only) the data properties of that entity. There are three data properties of note: unique identifier; arbitrary content; and, arbitrary attribute/valuesets (av-sets).

The property of unique identifier is trivial (though "details" such as construction of these identifiers may not be!)

The property of arbitrary content is clear enough: the content of an entity is an unstructured block of bytes. Of course, various services may interpret ("structure") this byte block in arbitrary ways. This is not the concern of the infrastructure.

The property of arbitrary av-sets is also straightforward. Attributes have names and sets of values. Attributes also have types, as in Construct [18], in which attribute types are one of: byte-block; offset; or reference. The type of an attribute constrains the values that may be placed in the value set of that attribute. Av-sets may also have further semantic constraints (e.g., the attribute named "color" must have exactly one value.)

Structure View and Properties. Our notion of structure view is again based on most existing structural computing work. The structure view of an entity highlights (only) the structure properties of that entity. There are three structure properties of note: unique identifier; arbitrary av-sets; and arbitrary reference sets.

The property of unique identifier is as described under the section on data properties.

The property of arbitrary av-sets is as described under the section on data properties, with the exception that there is no “reference” attribute type.

The property of arbitrary reference sets is also straightforward. As av-sets, reference sets have: a name; a set of references (identifiers of entities); and, a set of semantic constraints (e.g., this reference set must contain no more than 3 references). The semantic implied by reference here will be “pointing,” which seems to fit the idea of reference in most existing structural computing models, although there are clearly other choices (e.g., “inclusion.”)

Behavior View and Properties. Our notion of behavior view is uncharted territory with respect to existing structural computing work. The behavior view of an entity, as above, highlights only the behavioral properties of that entity. There are three behavior properties of note: unique identifier; image; and, location set.

The property of unique identifier is as described under the section on data properties.

The property of image is as described above under prototypic implementation of subleq – namely, a process memory space. There are no instruction operation codes in this image. Instead, there are only integers that may be potential parameters to subleq instructions.

The property of location set is a set of integers which are offsets into the image. These are to be interpreted as thread program counters. That is, each of the locations in this set (implicitly) indicate the three parameters to a subleq instruction.

Combining the Views. The basic DSB I entity, then, can be defined as follows:

$$dsb = \{id, C, R, L, A\}$$

where: id is a unique identifier; C (content) is an arbitrary byte block; R (refset) is a set of identifiers; L (locset) is a set of integers; and, A is an av-set.

The data view is defined as follows:

$$\begin{aligned} dsb_d &= \{id, C, A_d\} \\ A_d &= R \cup L \cup A \end{aligned}$$

The structure view is defined as follows:

$$\begin{aligned} dsb_s &= \{id, R, A_s\} \\ A_s &= C \cup L \cup A \end{aligned}$$

The behavior view is defined as follows:

$$\begin{aligned} dsb_b &= \{id, L, I_b\} \\ I_b &= C \cup R \cup A \end{aligned}$$

4.2 Implications of DSB I

What is behavior, then? Behavior, as data or structure, is merely a view to be imposed upon entities. When one views an entity as a behavior, one views the entity as an image – a set of parameters to be operated upon.

One advantage of using subleq as the basic behavior model is that any image (byte block) constitutes a viable program image. More complex behavior models will require instruction operation codes, which may become spoiled by manipulations of the image of an entity when viewed as data or structure. It is impossible to write (or generate) a syntactically incorrect subleq program. Whether or not semantics are preserved when entities are manipulated arbitrarily is another question entirely. However, this same problem exists in traditional data/structure syntheses, in which manipulations in the data view can potentially effect non-sensible structural semantics, even though all manipulations are syntactically permissible.

We have left many questions unanswered. Is subleq really the "correct" behavior model? Are pointing semantics for references really desirable? Should there be protections against manipulations of an entity in one view that have semantic implications in others? How could this be achieved? Is the notion of semantically meaningful even an issue that can or should be addressed in the infrastructure, or are semantics only a middleware layer concern? The model here allows for several different positions with respect to these issues. We will leave detailed discussion of these issues and the implications of the possible positions on them to another time.

Finally, one reason for the choice of "DSB I" as a name for the synthesis presented here is that many other choices may be made as to exactly how to synthesize data, structure, and behavior views (and even as to what constitutes these views.) We feel that this solution is one of many that we hope will be presented by various groups.

5 Related Fields

It is difficult to find directly related work to that presented here. Instead, in this section, we single out three fields to which there seem interesting connections. Two of these fields are not necessarily normally associated with structural computing, which makes them all the more enticing as places to look for new inspiration.

5.1 Connectionist Computing

By connectionist computing, we mean undertakings such as neural networks [14]. In such approaches to computing, computational engines consist of large numbers of simple computing entities that communicate with one another using very primitive messages. Often, messages consist only of one real number that acts as a signal strength. The computing entity may receive signals from many other "connected" entities, process this input in various ways, and then decide to send signals to various other "connected" entities. (What it means for entities to be "connected" here, or how such connections are built and/or destroyed, is not relevant to this discussion.)

There is a relatively straightforward implementation of neural networks using behavior views of DSB I. In terms of our subleq implementation, each computing entity (neuron) is a process, with one thread per connection. Each thread monitors a particular variable to which other threads in other processes may write values. Once the location monitored by a thread reaches a certain threshold, the thread writes some value to a location in some other process memory space.

The success of neural networks in solving relatively complex problems with extremely simple behavioral (computational) semantics should give us inspiration that the subleq model is not only theoretically complete, but may even be reasonably well suited real world tasks, given clever implementations. It might also inspire us to think in terms of very small and simple behaviors that can be combined into more complex systems.

5.2 Computational Theory

Tzagarakis [15] has introduced the notion of "structural completeness" analogous to "computational completeness" as described by Turing. Our behavior model is computationally complete, but is our structure model "structurally complete?" We've made no more progress answering this question than was initially described in [15], but we have introduced another possible line of attack in answering this question. Are computationally complete behavioral views that can manipulate structural view properties a sufficient means to effect structural completeness? We hope that Tzagarakis and others might use the notion of behavior view to make headway on the structural completeness problem.

5.3 Process-Oriented Hypertext

One of the very first research hypertext systems was PROXHY[4] (later redubbed SP0/HB0.) PROXHY is an acronym that stands for "process-oriented hypertext" – a very computationally driven approach to viewing hypertext systems. In PROXHY (and later formalized in the HRL model of hypertext [5]), links and anchors are not objects (as in most other models and systems) but rather arbitrary processes. Another example of a highly computational approach to viewing hypertext systems is Multicard [12], in which links serve as communication pathways for events between hypermedia objects.

Though the SPn/HBn work "lives on" in Construct [18], the codebase successor to the HRL systems, the notion of process-oriented hypertext seems to have faded out of focus. Indeed, Construct implements a community "standard" model of (navigational) hypertext in which links and anchors are again simply (non-computational) objects stored in an infrastructure (though arbitrary, poorly defined, and effectively second-class "behaviors" may execute over these objects.)

Perhaps with computation firmly re-rooted in the structural computing models, it is time to take a look back at the early HRL models and see how computation was leveraged, and examine whether or not structural computing systems base on a data/structure/behavior synthesis (DSB I or otherwise) can capture those same advantages.

6 Conclusions and Future Work

In this paper, we started out making a claim analogous to that made at the initial proposal of structural computing: namely, behavior was (oddly enough) a second-class entity in a field ostensibly about (structural) computing. How can behavior be raised into first-class status? We formed a proposal for a solution to this problem again using analogous means to the ones used in the initial structural computing formulation: we tried to synthesize the existing infrastructure abstraction (already a data/structure synthesis) and a (simple but complete) behavior abstraction into a synthetic atom that may be viewed as data, structure, or behavior, as appropriate.

It is much too early to have extensive results on the usefulness of this project, but there are interesting questions raised by this experiment up front. We feel that the method of synthesizing first-class abstractions into a basic atomic entity has shown to be useful in "traditional" structural computing work, and hope that this community will find analogous benefits in our proposal.

References

1. Anderson, K. M. and Sherba, S. A. 2001. Using structural computing to support information integration. In [11].
2. Anderson, K. M. 2000. Structural computing requirements for the transformations of structures and behaviors. In [10].
3. Hicks, D. L. 2000. Structural computing: evolutionary or revolutionary? In [10].
4. Kacmar, C. J. and Leggett, J. J. 1991. PROXHY: a process-oriented extensible hypertext architecture. *ACM TOIS (9)*(4), ACM Press, NY, 399-419.
5. Leggett, J. J. and Schnase, J. L. 1994. Viewing Dexter with open eyes. *Comm. ACM 37*(2), ACM Press, NY, 76-86.
6. Millard, D. E. and Davis, H. C. 2000. Navigating spaces: the semantics of cross-domain interoperability. In [10].
7. Nürnberg, P. J. (ed.) *Proceedings of the First Workshop on Structural Computing (SC1)*, Technical Report CS-AUE-99-04, Department of Computer Science, Aalborg University Esbjerg. Available at `http://cs.aue.auc.dk/publications/tech.html`.
8. Nürnberg, P. J., Leggett, J. J., Schneider, E. R. 1997. As we should have thought. *Proceedings of the Eighth ACM Conference on Hypertext and Hypermedia (HT 97)* (Southampton, UK, Apr).
9. Nürnberg, P. J., Leggett, J. J., Schneider, E. R. and Schnase, J. L. 1996. Hypermedia operating systems: A new paradigm for computing. *Proceedings of the Seventh ACM Conference on Hypertext and Hypermedia (HT 96)* (Washington, DC, US, Mar).

10. Reich, S. and Anderson, K. M. (ed.) *Open Hypermedia Systems and Structural Computing: Proceedings of the Sixth Workshop on Open Hypermedia Systems (OHS6) and the Second Workshop on Structural Computing (SC2)*, Lecture Notes in Computer Science vol. 1903, Springer-Verlag: Heidelberg.
11. Reich, S., Tzagarakis, M., M., and De Bra, P. M. E. (ed.) *Hypermedia: Openness, Structural Awareness, and Adaptivity: Proceedings of the Seventh Workshop on Open Hypermedia Systems (OHS7), the Third Workshop on Structural Computing (SC3) and the Third Workshop on Adaptive Hypermedia (AH-3)*, Lecture Notes in Computer Science vol. 2266, Springer-Verlag: Heidelberg.
12. Rizk, A. and Sauter, L. 1992. Multicard: an open hypermedia system. *Proc. ECHT 92* (Milano, IT, Oct).
13. Rosenberg, J. 1999. A hypertextuality of arbitrary structure: a writer's point of view. In [7].
14. Rumelhart, D. E., Widrow, B. and Lehr, M. A. 1994. The basic ideas in neural networks. *Comm. ACM 37*(3), ACM Press, New York, 87-92.
15. Tzagarakis, M. 2003. personal communication.
16. Vaitis, M., Papadopoulos, A., Tzagarakis, M. and Christodoulakis, D. 2000. Towards structure specification for open hypermedia systems. In [10].
17. Wiil, U. K. and Hicks, D. L. 2001. Providing structural computing services on the World Wide Web. In [11].
18. Wiil, U., Hicks, D., Nürnberg, P. 2001. Multiple open services: a new approach to service provision in open hypermedia systems. *Proceedings of the Twelfth ACM Conference on Hypertext and Hypermedia (HT 01)* (Århus, DK, Aug).
19. Wikipedia – OISC. `http://www.wikipedia.org/wiki/OISC`.

A Meta-modeling Approach to Ontological Engineering: DL-Workbench Platform

Mikhail Kazakov[1,2] and Habib Abdulrab[2]

[1] Research division, Open Cascade S.A. 4, rue Rene Razel, 91400, Saclay, France
mikhail.kazakov@opencascade.com
[2] PSI laboratoire, INSA de Rouen, BP 8, 76131, Mont Saint Aignan, France
abdulrab@insa-rouen.fr

Abstract. Nowadays many ontological editors are available. However, several specific requirements forced us to search for new principles and to develop a new ontology edition platform called DL-workbench. The platform combines research experience of model-based software architectures and knowledge based engineering. We applied the metainformatics principles in order to satisfy growing requirements of our application domain. DL-workbench contains three main modules. First module defines a meta-model for description of ontological formalisms. It provides an API that allows management of ontological entities, containers of entities, reasoner connection and many other features that are useful when one needs to use an ontological model within its project. A second module of DL-workbench is a formalism-independent user interface for edition of meta-model based ontologies. This module uses the meta-model and is implemented as a plug-in to the IBM Eclipse platform. A third module defines the SHIQ description logic formalism using the meta-model. The third module provides also customization for the user interface of the second module (such as images, literal names, etc.). DL-workbench pays much attention to the management of complex logical equations as well as management of many ontologies within a project. DL-workbench allows easy integration of ontological model with other data inside one standalone or distributed application. DL-workbench can be used both as the ontological editor and as an ontology manipulation platform integrated with other tools and environments. This paper describes our motivation for creation of DL-workbench, implemented features and lessons learned from implementation of ontological editor.

1 Introduction

Metainformatics is an extremely important direction of computer science research. Exchange of research results from different communities often leads to advancements in academic and applied research. Such an exchange leaded us to creation of a software framework that combines principles of model-based software architectures (meta-modeling approach) and research in ontological engineering. Hereinafter we present the DL-workbench platform that implements our research results in building of ontological frameworks.

Ontologies have become an increasingly important research topic. This is a result of their usefulness in many application domains (software engineering, databases, medical domain, conceptual modeling, etc.) and the role they will play in the devel-

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 17–33, 2004.

opment of emerging Semantic Web activity. It is clear also, that development and manipulation of ontologies have to be supported by corresponding software tools (annotation tools, editors, reasoners, etc.) and standards (OWL [9], etc.). However, most of the tools are focused on the ontological engineering, but not on architectural aspects of ontological platforms. With DL-workbench we tried to fix this gap and present an open source tool that provides a wide research community with a convenient utility for integration of ontological functions within their environments.

Nowadays many application domains are looking for use of ontologies. And each domain, that uses ontologies, sets its own requirements for tools. One of such domains is a software integration domain which is a subset of software engineering area. The main challenge of integration is: how to make working together software entities that were not initially created to work with each other. The authors of this paper are currently working on the topic of semi-automated integration of various numerical simulation multi-physics solvers [2] within a distributed environment. Here the use of ontologies (as formal description models) helps to share the same view on specification of solvers that come from different vendors. Reasoning procedure allows retrieving the needed software configuration. Integration topic is out of the scope of this article and will be published in a separated paper. Some preliminary results were given in [19].

Since we use ontological models in our specific domain, we need to choose a formalism that will be used for creation of these models. In addition we need tools that support this formalism and provide us with convenient user interface (creation of ontologies and reasoning). Finally we need an API that allows us to integrate all these tools with software solutions, specific to the integration domain (code generators, etc.). Taking into account the research origin of our work, we have formulated a set of requirements for an ontological tool that would be convenient when working within a specific application domain:

- Full support of at least one ontological formalism. Several formalisms are preferred for research purpose. During the work on our research project dedicated to semi-automated integration we experimented with several ontological formalism and at the moment of DL-workbench creation we were not sure, whether description logics are appropriate or not for our research. The next requirement is coming from the same consideration.
- The ability to switch among ontological formalisms for research purposes.
- Convenient user interface to work with complex logical expressions (with ability to modify the structure of expressions via the graphical user interface). While trying several ontological editors, we found that most of the ontological editors don’t implement expression editors. Those who implement them were not convenient from our point of view.
- Presence of reasoner connection for implemented formalisms. Within our research we have to work with complex logical axioms and ontological models. These require the presence of reasoner both for validation of ontologies and for performing of additional reasoning tasks that are dedicated to semi-automated integration of software components.
- Ability to integrate that tool into a specific domain environment (API). Our research is dedicated to integration environments that already exist as prerequisite. The next requirement is coming from the same origin.

- Ability to manipulate ontologies and their elements from specific domain environment (API). We need that in order to merge several ontological formalisms within the same environment. For example the ontological data is merged with specifications of Java interfaces.
- Ability to work with structured "ontological projects" but not with "files". Following our methodology of semi-automated integration we need to work with several separated ontological files at the same time. Managing of these files within the "project"-like environment seems to be an acceptable solution. Most of the people in software engineering who will use our products are familiar with this concept.
- Fast and portable user interface. This and below-listed requirements are coming from the considerations of having good practices of software development.
- Convenient API based on design patterns.
- Presence of extension points.

We noticed that the same requirements arise often within the variety of other domains where ontologies are used (manufacturing engineering, configuration, etc.). One can surely mention other important requirements such as support of collaborative work, versioning and many others. We surely admit their importance however here we present the list of our major requirements.

We have studied the state of the art of ontology creation/edition tools and programming interfaces. However we did not find any tool that could comply with our requirements and be convenient for our needs. Thus the decision to create own platform was taken. The name of the platform is DL-workbench (short of Description Logic Workbench, since SHIQ description logic [7] is the main formalism that is used). DL-workbench is published now under open source license (http://www.opencascade.org/dl-workbench).

The DL-workbench is a meta-model based ontological edition platform. The extension and integration APIs are open and documented. Meta-model allows us to switch easily between ontological formalisms (such as description logics [1], first order logic, F-logic [15] and others) and gives the ability of easy and extensible integration with other environments. DL-workbench is implemented as a set of IBM Eclipse [10] plug-ins using Java language. That fact allows seamless integration of DL-workbench with many other software engineering tools (IBM WebSphere, Rational XDE, etc.).

We consider the metainformatics approach to be extremely useful for advancement of computer science research. People coming from different domains (Artificial Intelligence, Constraint programming and Software engineering) came to an idea of a tool that combines approaches from these domains (ontological models, meta-modeling approach, using of design patterns, pre-post conditions and invariants processing engine, etc.). That leaded to creation of the modern tool that is highly extensible and satisfies all the requirements of our project. We hope that our experience in creating of software tools for manipulation of ontologies (and DL-workbench itself) will be useful for the wide range of researchers in computer science who use ontologies in their work. This paper describes a meta-model approach that was used for creation of Dl-workbench. First we give the concept of DL-workbench and our motivations. Further, we describe the meta-model kernel and other modules. At the end we share some of lessons learned during creation of this tool.

2 DL-Workbench Conception

We have explored many existing tools and API's for manipulation and edition of ontologies (see [20] for ontology editors survey). Absence of a product that satisfies all our needs and communicating on research forums forced us to develop a new generic platform – DL-workbench.

The next list of principles was formulated and it constitutes the main concept of DL-workbench:

- DL-workbench is based on a meta-model that is capable to describe the structure of ontological formalisms and third-party data that can be used within a specific domain.
- DL-workbench has a modular (plug-in based) architecture with clearly specified dependencies among modules. Each sub-tree of modules (within the dependency tree) can be used as a separate module.
- Main processing module of DL-workbench is based only on the meta-model. It does not depend on any of specific formalisms. This module implements features that can be implemented using only generic meta-model (persistence skeleton, reasoning skeleton, tracking of changes, transactions[1], lifecycle of instances and many others). Only top level modules (leafs of dependency tree) are formalism-specific.
- Each generic module provides a set of documented extension points that can be used by other modules / software to customize the platform.
- The work with ontologies is performed using a notion of a project. A project here is a structured set of ontological files (ontological resources) and other domain specific data if needed.
- DL-workbench defines an internal data model and "UI-ready" data model. That allows organization of different views on the same data (i.e. project view, namespace view, taxonomy view).
- DL-workbench supports manipulation/edition of complex expressions and axioms. In many application domains, the complete and "reasoning-able" ontologies require the use of logical expressions and axioms.
- DL-workbench provides a structured and documented API over all above mentioned features.
- DL-workbench provides the user interface of an ontological editor.

We strongly believe that an ontological manipulation platform shall be based on these principles to be interoperable and useful for researchers, software architects and end users. DL-workbench source code is public and we hope it will be useful for vendors of ontology edition tools.

DL-workbench can be viewed both as a meta-model based platform for creating ontology-manipulation tools and as an ontological editor that supports SHIQ description logic [7] (the meta-model is used to describe the structure of SHIQ logic formalism). DL-workbench uses DAML+OIL[2] [8] as persistent format and Racer [11] as DL reasoner. SHIQ was chosen due to the following main reasons:

[1] Transaction support is not implemented in the current version of DL-workbench

[2] We consider using the OWL language [9] instead of DAML+OIL as soon as OWL parsers will appear on the market.

- Description logics were taken as the most appropriate language for formal specifications within our domain [19].
- SHIQ is a very expressive decidable description logic that has implemented reasoners.
- SHIQ and DAML+OIL are supported by semantic web community.

The use of meta-modeling approach to creation of tools was successfully used before for products such as IBM WebSphere, Rational Rose and many others. DL-workbench is implemented as a set of plug-ins to IBM Eclipse platform [10]. Eclipse is an emerging open source Java-based environment for creation of project-based tools. Eclipse implements its own portable UI widget library (SWT [10]) that uses native OS calls and is much faster than SUN Swing library [13]. Fig. 1 shows the general structure of the DL-workbench.

Each arrow shows the use of API of the corresponded module. Below we give details of the meta-model kernel and some interesting aspects of DL-workbench.

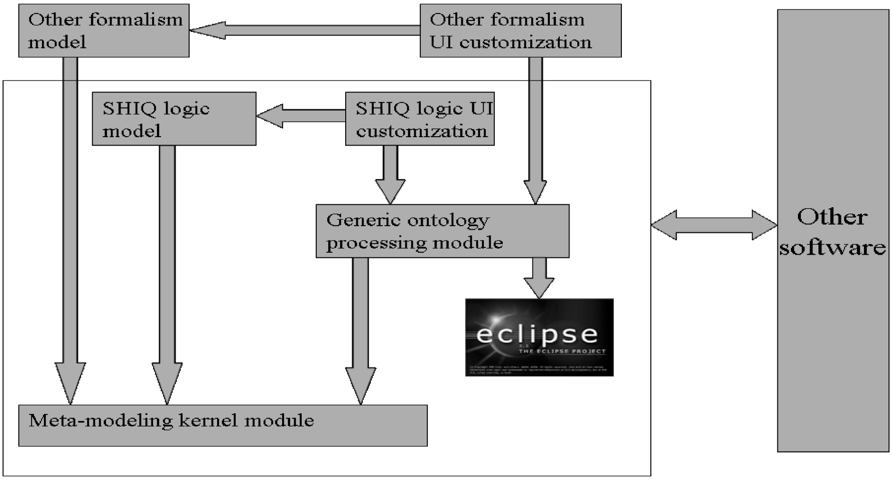

Other formalism model
Other formalism UI customization
SHIQ logic model
SHIQ logic UI customization
Generic ontology processing module
eclipse
THE ECLIPSE PROJECT
Meta-modeling kernel module
Other software

Fig. 1. General structure of DL-workbench.

3 Meta-model

The major advantage of the DL-workbench is a meta-model. It allows easy-to-use definition of entities and relations of an ontological formalism that has to be used within the workbench.

The meta-model is implemented as a set of Java interfaces for the convenience of use from programming environments. Its UML representation is presented below on the Fig. 2.

In software engineering, the term “meta-model” refers to a language that is used to define structure of “models”. These “models” define structures of final application data (i.e. “models” define meta-data). We use meta-model as a language for description of ontological formalisms by specifying their elements, structure and invariants (invariants have to be satisfied when instances of these elements are created or modi-

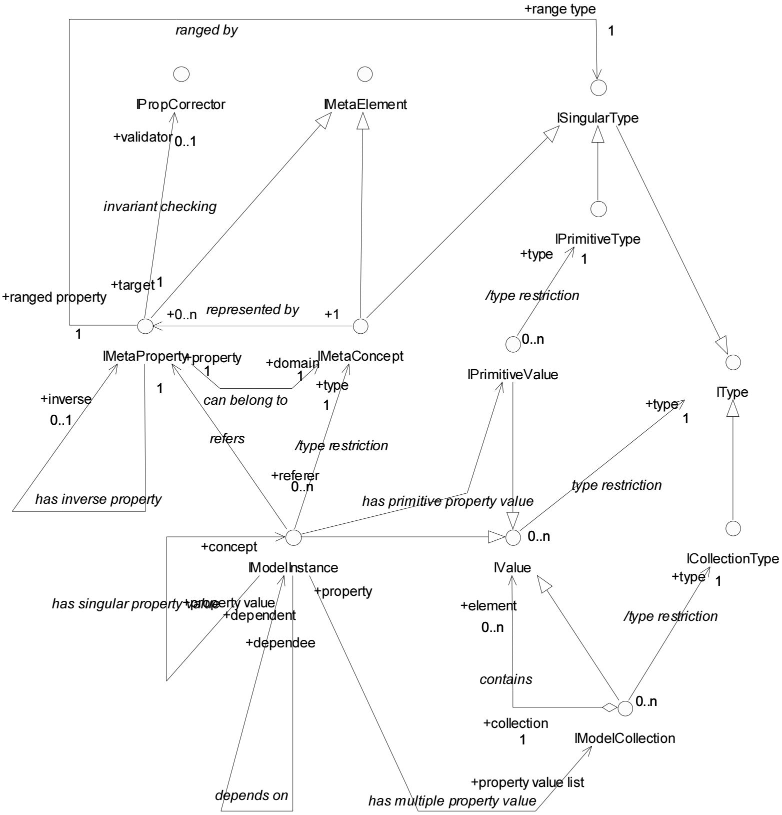


Fig. 2. UML class diagram of main interfaces of the DL-workbench meta-model.

fied). For example: one needs to work with a simple propositional logic that includes notions of “atomic proposition” and “composite proposition” expressed via logical expression with conjunction, disjunction and negation operands. Let us imagine that he needs to edit and manipulate models, based on the given logic. In this case he must implement the user interface, persistence, objects lifecycle, etc. Meta-model based approach can provide that person with an extensible description mechanism that enables reusing of existing features (user interface, object lifecycle, etc.) and facilitate modular integration with third-party solutions.

Our meta-model is implemented as a separate module and is independent from Eclipse or any other tool. The module is very light and can be used in a network environment. The main goal of the module is to achieve the maximal level of independence from specific formalism and to enable implementing generic software features (object lifecycle, transactions, etc.). The module also helps achieving the interoperability among different tools by sharing the same meta-model.

The meta-model is basically intended for specifying structural models. Semantics of the meta-model were inspired by Description Logics [1].We tried to keep the meta-model as simple as possible, but powerful enough for many possible needs.

The main element of meta-model is a meta-concept (IMetaConcept interface). It represents any typed element with a set of properties. For example: "Expression", "proposition", "atomic proposition", "logical operand" are the instances of meta-concept. Each meta-concept has a meta-name that is represented in a form of java interface. The taxonomy of these Java interfaces defines the taxonomy (i.e. subsumption relationships) of corresponded meta-concept instances. For example: an "atomic proposition" is a "proposition" and an "expression" with atomic propositions is also a "proposition". We can define three java interfaces (IProposition, IAtimicProposition extends IProposition, IExpression extends IProposition). Their taxonomy represents the taxonomy of our concepts. The use of Java inheritance model to define subsumption is an important design choice. Our experience shows, that it simplifies creation of code that is based on the meta-model, reuses Java language reflection features and makes the code much more readable and understandable.

Another important element of a meta-model is a meta-property (IMetaProperty interface). A meta-property represents a property that can potentially restrict a meta-concept (similar to description logic semantics of "property"). For example, the "atomic proposition" concept has "name" property; an "expression" concept has "operand", "left part" and "right part" properties. Meta-property also has a meta-name that is represented in the same way as for meta-concepts (i.e. in the form of Java interfaces). Each meta-concept instance may have a set of meta-property instances as its definition. An instance of meta-concept that inherits other meta-concepts also takes all their properties. Generally, the most specific meta-concept is restricted by the transitive closure of all the properties.

Any meta-property has the notion of domain restriction, range restriction, inverse property restriction and transitivity. Semantics of all these notions (meta-concept, meta-property, domain, range, etc.) are very close to semantics of corresponded elements from description logics. "Domain restriction" of a property is an instance of meta-concept that defines the most generic type that can accept this instance of meta-property as its property. Domain restriction is introduced only for model checking purpose (in case, if this restriction is violated while creation of a model, the exception is raised). Range restriction defines the types of elements that can be accepted by the property. It can have a "singular" or "collection" type (type system description is given below). "Inverse" is a meta-property that will be automatically connected with this property by values. For instance "left part" property of "expression" may have the range of "proposition" type and can have the inverse property "belongs to left part of" of "proposition" concept. When the "left part" value of an "expression" is set to a specific "proposition" instance, the "expression" will be automatically added to the "belongs to left part of" collection of this "proposition". Presence of such a property within the meta-model is very convenient, when a model requires computable (derived) properties (for example, the "sameAs" property, or "conceptHasInstance" / "superClassOfInstance" pair within SHIQ description logic). In such cases the notion of inverse property eliminates the need in additional run-time computation. However, the redundant information is introduced into the final model.

The notion of transitive property is similar to the semantic of transitivity in description logics. Transitivity is never used in DL-workbench modules. However this information can be useful when writing the formalism-specific module. Other description logic semantics (unambiguous, unique properties) are not useful for creating of models and automated processing. The meta-property is functional by its definition and uniqueness is given by the cardinality of range restriction (a meta-concept takes always properties with predefined range).

In addition to these semantics, each meta-property can be augmented with a pre-post processing handler for checking the invariant conditions of given property values. For example: the "name" property of the "proposition" concept has a "String" type. It must be complaint with the URI specification (i.e. no spaces, quotes, etc). An invariant handler that does such check can be written for the "name" property. Each string value that is set as "name" of any "proposition" will be automatically passed through this handler. Another example is a check, that "expression" with "negation" operand has only one parameter.

The meta-model includes a simple type system. Types are used to define meta-property range restrictions and to type instances of meta-model elements. Type system includes singular types and collection types. A singular type can be primitive (String, Boolean, Integer, Float, and Enumeration) or a meta-concept instance. Collection type is defined by the type of its elements (any other type).

In order to specify some logical model (formalism), meta-concept and meta-property classes must be instantiated. A special meta-model factory is implemented within DL-workbench to facilitate this task. It is worth to mention, that many formalisms can exist at the same time, can share their definitions and can be searched for specific meta-concepts and meta-properties.

Here it's necessary to mention, that three meta-modeling levels exist. `IMetaConcept` interface has the semantic of "meta-concept", its Java instance is a specific instance of the "meta-concept" (model concept) and `IModelInstance` interface specify an instance of the model concept (i.e. model instance). In order to instantiate the formalism, the notion of value is defined (IValue interface). Every primitive type and collection has its value object (`IPrimitiveValue, IModelCollection`). Every instance of meta-concept within some formalism can have many model instances represented by `IModelInstance` interface.

Each model instance has its type represented by a meta-concept instance and a set of values according to properties of the model concept. For example: a model instance of "expression" meta-concept can be created with three values of their properties:

- "left part"
  - model instance of type : "atomic proposition"
    - "name" = "MARY" : String
- type "operand"
  - model instance of type : "conjunction"
- type "right part"
  - model instance : type "atomic proposition"
    - "name" = "PETER": String

As we can see the instances of the above mentioned simple propositional formalism are expressed in terms of meta-model. The structure of the ontological formalism can be easily traversed by any application. Consequently, an ontological editor that supports only the notion of (concept – property – value) triple can be easily implemented. That is extremely important for specific application domains, where the ontological information must be included within some other application.

Use of Java reflection mechanism is important here for the clarity of code. The meta-names are used to access the value of specific property of an instance of meta-concept by its clients. Java type checking mechanism together with domain/range restrictions on properties allows model checking and simplifies debugging of software.

It is necessary to mention, however, that ontological reasoning and persistence may be supported only by means of generic interfaces within the meta-model. In order to connect the reasoner, it's necessary to take into account the structure of formalism. The same rule applies for persistent formats. DL-workbench meta-model kernel defines the generic interfaces for connecting the reasoner, passing models to/from reasoner, querying the reasoner, persisting models.

The meta-model can be used not only for description of ontological formalisms, but also for description of other data formats that are needed for specific application domain. For example, we use description of Java interfaces expressed with the same meta-model within the software integration domain. Many structural data formalisms can be easily expressed in the presented meta-model.

We don't support concrete domains or cardinality restrictions within the meta-model (i.e. we can not specify, that range of "left part" of "expression" must have at least one instance). However, invariant checking feature allows introducing such checks by the mean of coding. We didn't have the goal to create the good-for-all meta-language. The decision to use coding simplifies the meta-model and its processing.

The last element of meta-model is a "Container". Container is a collection of model instances or other containers that supports basic operations of addition, removal, iteration, checking of containment and size request. Each container can be restricted by types of values it can store. The separated notion of containers is very important while working with ontologies. Creation of dynamic groups of elements is useful when sending information to reasoner, saving sub-ontology, classifying elements, etc (for instance, description logics ABox and TBox can be transparently implemented).

The meta-model kernel has a small abstract model expressed by means of meta-model. We believe that this model is useful for expression of many different formalisms. Its meta-concepts and meta-properties (that are instances of IMetaConcept and IMetaProperty consequently) are described below:

- Serializeable container – a specific type of container that is ready for serialization into any formalism-specific format (the converter has to be registered). Every such container is accessible from a special repository.
- Namespace container – a specific type of container that represents all elements from one namespace (i.e. having one namespace URI). Most of the ontological formalisms support the notion of namespaces thus we've decided, that this notion is generic for many tools. Every namespace is accessible via special repository.
- "Model instance holder" concept – the most generic concept for every other concept that is using this structure.
- "Serializeable element" concept – an element that is considered to be persisted. Invariant checker of such element connects it with its serializeable container that is bound to a physical source (file, URL, etc.).

- "Named element" concept – an element that has literal name and description. Inherits "model instance holder" concept. Invariant checker converts automatically any name to URI-complaint format. Two properties are defined:
  - "name" : String
  - "description": String
- "Namespaced element" concept – any element that has a namespace. This concept inherits "Named element" concept thus inheriting all its properties. Invariant checker of such element automatically adds itself into the namespace container according the value of "namespace URI" literal property. One property is defiled:
  - "namespace URI" : String
- "Operand" concept – a logical operand. Basic operands are given such as: AND, OR, NOT, XOR, IMPLIES, EQUALS, FORALL, EXIST. One property is defined:
  - "operand": Enumeration
- "Expression" element – a generic structure of logical expression. Most of the ontological formalisms support expressions. The presence of such element on this level allows us to implement generic expression editor. Has three properties:
  - "Logical operand": Operand
  - "Number of parameters": Integer
  - "Parameters": Collection of Model Instance Holders

With the simple example we have shown principles of working with meta-model and creating of models. The DL-workbench meta-model documentation [5] may be consulted for further information.

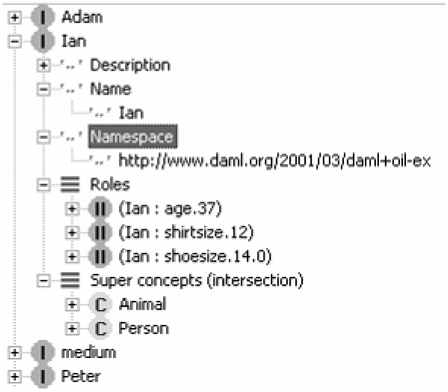
4 Processing Module

A generic ontology processing module implements manipulation and edition of ontologies. This module is integrated with Eclipse workspace and depends only on the meta-model module. Eclipse framework provides us with the project-oriented workspace. The framework enables transparent connection to many environments of software integration domain (IBM Web Sphere J2EE and WDSL environment, Eclipse Java development framework, some UML tools, etc.).

The processing module implements the "view" and "controller" concepts according to the Model-View-Controller paradigm. Here the "model" concept is represented by a formalism that is an instance of the meta-model. "Controller" can be viewer as the UI operations. "View" data model is built on top of meta-model and enables many representations for specified formalism. An ontological project can be viewed by default as:

- a structured set of persisted ontologies (files)
- a set of namespaces (in case if formalism supports namespace)
- each element of some formalism is presented by a tree including properties of corresponded meta-elements and their typed values

Fig. 3. illustrates one of the views (tree of properties) on the example of SHIQ logic.



```
⊞ I Adam
⊟ I Ian
   ⊞ '..' Description
   ⊟ '..' Name
        '..' Ian
   ⊟ '..' Namespace
        '..' http://www.daml.org/2001/03/daml+oil-ex
   ⊟ ≡ Roles
        ⊞ II (Ian : age.37)
        ⊞ II (Ian : shirtsize.12)
        ⊞ II (Ian : shoesize.14.0)
   ⊟ ≡ Super concepts (intersection)
        ⊞ C Animal
        ⊞ C Person
⊞ I medium
⊞ I Peter
```

Fig. 3. Properties of "UnaryIndividual" element. Daml+oil-ex.daml ontology is used.

The processing module defines all the generic UI operations for lifecycle of model instances independently of the formalism used. Processing module generates user interface controls following the structure of a given model instance. For example: the "expression" meta-concept instance has two properties of type "proposition" and one property of type enumeration. In this case, when a user asks for edition of an instance of "expression", three groups of controls will be generated independently on the end-user semantics of "expression". This principle works for any operation within the module. User interface elements are created only once for each type of elements and cashed to reduce unnecessary OS interactions. In order to achieve the end user presentation shown on Fig. 3, other module must specify the formalism itself and its UI resources (icons, names, order, etc.). Several concurrent/joint formalisms can be also supported. Further details of implementation, extension points and API can be found in [5].

5 SHIQ Module

The SHIQ module consists of two parts. First part defines a model of SHIQ formalism that is based on the meta-model, implements DAML+OIL reader and writer and defines DIG interface connection for the solver. The implementation of SHIQ formalism is described in [5] and is not repeated in this paper. The formalism contains 20 instances of meta-concepts {UnaryExpression, Restriction, Predicate, Axiom (left predicate, operand, right predicate), CompositeAxiom (list of equivalents, list of disjoints), UnaryPredicate, BinaryPredivate, Datatype, Concept, DatatypeProeprty, ObjectProperty, DatatypeValue, UnaryIndividual (concept instance), SetOfIndividuals(for oneOf restriction), BinaryINdividual(property instance), ExpressionOperand, RestrictionQuantifier, RoleFiller (UnaryIndividual or DatatypeValue) and BinaryExpression}. The SHIQ formalism contains also 38 instances of meta-properties that restrict above presented instances meta-concepts.

DL workbench supports the complete list of SHIQ semantics [7]. It also has declarative support of concrete domains (i.e. no operations such as comparison, range checking and others are supported within meta-model for the concrete domains). Our application domain doesn’t require concrete domains; moreover they are not supported by used reasoners. Nevertheless, the existing formalism can be easily extended to support concrete domains.

Second part of SHIQ module customizes user interface for use with the processing module. Literal names for every SHIQ concept and property are listed. Icons, representing concepts and properties are also provided by this part. The list of editable properties and their order is defined for each meta-concept instance (for example SHIQ DatatypePropety concept will have next properties displayed for edition in the next order: Namespace, Name, Description, Uniqueness flag, Range, Domain).

The taxonomy view is implemented in the SHIQ module. Selecting of any SHIQ “concept” or “object property” concepts within the Eclipse workspace causes the dynamic building of the taxonomy tree for this concept/property. The view is shown by default on the right edge of the Eclipse window within the DL-workbench perspective. The taxonomy is dynamically rebuilt using “subClassOf” and “sameAs” properties of the SHIQ “concept”/“properties”. SHIQ module (as any formalism specific module) also specifies a set of Eclipse extensions: association of *.daml files with the plug-in, association of specific icons with menu items and others UI features. Integration with Eclipse is described in DL-workbench documentation [5].

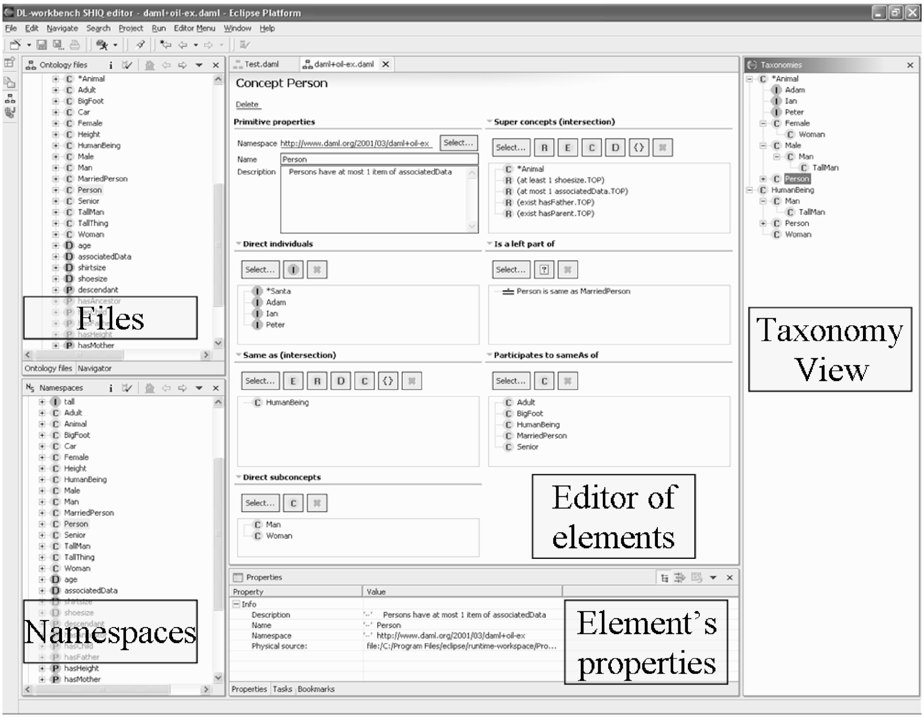
As we can see, the inclusion of specific formalism and customization of the user interface represent a very small part of the ontological editor code. For example, the F-logic formalism was defined as an example during several hours using the same meta-model and benefits all the editor features (F-logic module is not published with DL-workbench due to incompleteness and absence of persistent format connection).

Racer reasoner [11] is used via DIG interface [4]. We choose Racer due to its support of ABox reasoning. However, the DL-workbench itself uses a reasoner only for satisfability and subsumption checks, thus FaCT [12] or any other DIG-complaint reasoner can be used. DAML+OIL reading support is done with the help of Jena DAML parser [17]. DAML+OIL writing was done via Xerces XML parser [16]. Since DAML+OIL is a superset of RDFS, any RDFS file can be read as well by DL-workbench ontological editor.

6 Ontological Editor User Interface

The work with ontological editor is described in the end user documentation in [5]. In this chapter we show some screenshots of the user interface.

Since DL-workbench is implemented as a set of Eclipse plug-ins, it follows the general conception of Eclipse workbench. A user works with the notion of project by adding and removing files form it. When adding a DAML+OIL file (or other file, if some other formalisms are defined), DL-workbench parses it and loads ontological elements into the framework. After that every element is accessible for modification. Fig. 4. shows the main window during edition of an element. Thee views are defined for the SHIQ editor. “File views” (see Fig. 4) displays project structure in terms of files and contained elements. “Namespace view” shows the same ontological elements, but arranged with respect to their namespace. “Taxonomy view” displays the selected element according its parents/children.

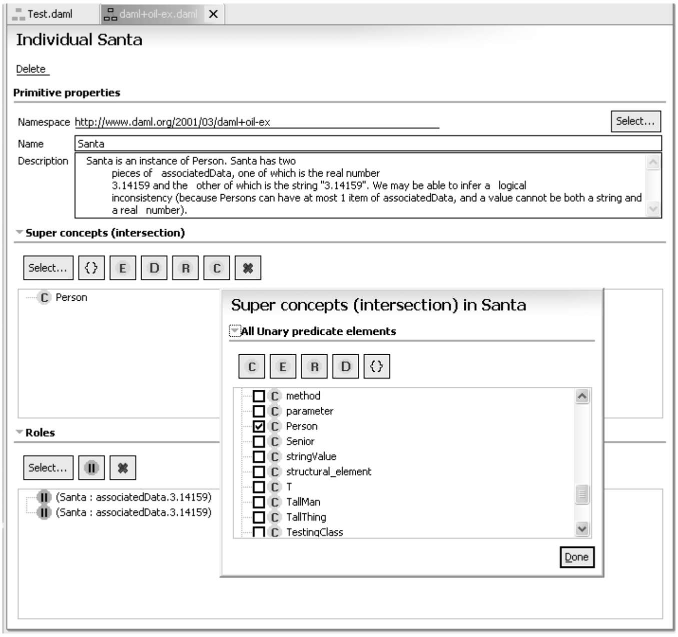


Namespace http://www.daml.org/2001/03/daml+oil-ex
Name Person
Description Persons have at most 1 item of associatedData

Property	Value
Info	
Description	Persons have at most 1 item of associatedData
Name	Person
Namespace	http://www.daml.org/2001/03/daml+oil-ex
Physical source:	file:/C:/Program Files/eclipse/runtime-workspace/Pro...

Fig. 4. General look and feel of DL-workbench. Daml+oil-ex.daml ontology is used.

Property panel (at the bottom of the screen) displays the primitive properties of selected element (such as name, namespace URI, file, literal description, etc.) Editor panel takes the biggest area of the screen and allows edition of properties of chosen instance. Fig. 4 illustrates the process of edition of "Person" concept from the standard daml+oil-ex.daml ontology. Fiture 5 shows in details the editor panel with "Santa" individual from the same ontology. On this picture we can see that "Santa" is a SHIQ individual that has 5 properties: Namespace URI, name, description, intersection of super concepts and roles. The "Super concepts…" panel displayed on top of the window allows selection of existing concepts for "super concepts" property of Santa individual or creation of new ones. The list of meta-elements that can be created ("C", "E", "R", etc. buttons) is built using the range of "Super concepts" meta-property instance. The small window on top of the editor panel shows all the possible elements that can be used for edition of one of the properties of ontological element. An end-user can either select existing elements from a list or can create new elements on fly by clicking appropriate button and typing of names of the new element. The list of existing elements is built by taking all the possible elements of a type that is compatible with the type accepted by the property being edited. The list of concepts is taken by traversing of all concepts within all namespaces loaded within current DL-workbench session. In this case the elements can be shared among files and DL-workbench tracks these links within the structure of projects. More details on the user interface and use of DL-workbench as ontological editor can be taken from [5]. The product itself and its source code can be downloaded following link in [5].



Individual Santa

Primitive properties

Namespace http://www.daml.org/2001/03/daml+oil-ex

Name Santa

Description Santa is an instance of Person. Santa has two pieces of associatedData, one of which is the real number 3.14159 and the other of which is the string "3.14159". We may be able to infer a logical inconsistency (because Persons can have at most 1 item of associatedData, and a value cannot be both a string and a real number).

Super concepts (intersection)

C Person

Super concepts (intersection) in Santa

All Unary predicate elements

- □ C method
- □ C parameter
- ☑ C Person
- □ C Senior
- □ C stringValue
- □ C structural_element
- □ C T
- □ C TallMan
- □ C TallThing
- □ C TestingClass

Roles

(Santa : associatedData.3.14159)
(Santa : associatedData.3.14159)

Fig. 5. Example of editor UI for an instance of UnaryIndividual concept of SHIQ model.

7 Lessons Learned

In this section we'd like to indicate some positive experience and observations that were received during the creation of DL-workbench and use of ontologies for a specific application domain.

Axioms and logical expression are extremely important for creating of complete and reasoning-ready ontologies. However, it's extremely difficult to have a convenient user interface (GUI) for their edition. We've implemented our own GUI of expression editor; however we strongly believe that some deep research must be conducted on ergonomics of expression editor. It can be something between text editor with dynamic compilation and GUI based editor that allows choosing needed elements from lists.

From our point of view, working with ontologies must follow the project-oriented paradigm. It's hard to imagine a real industrial ontology that is saved in one file and has no references to other files. The use of URI as a physical location of imported ontology is not always suited for industrial use due to possible unavailability of some URI at some time. Here the notion of project as a complete set of needed ontological resources can facilitate manipulation of ontologies. It can clearly separate a physical structure of files from logical structure of ontologies (i.e. namespaces, taxonomies).

The ability to have many different views (by namespaces, by taxonomies, by files, graphical view and many others) on the same ontological structure helps a lot in many real cases.

Presence of meta-model for implementation of ontological formalism and connection with other data structures is very important. Above we said many things about these benefits.

Support of several formalisms and several GUI views is extremely important since it allows creation of different views for different groups of users on the same domain data and its ontological semantics. For example the same ontology may be presented by two formalisms with different expressivity to different groups of people.

We found it useful to introduce several macro-semantics into the SHIQ editor that are computed from basic SHIQ semantics:

- XOR, IMPLIES and EQUALS logical operators can be easily introduced into any expression within SHIQ model. These operands are easily convertible into AND/OR/NOT sequences and easily extractable from such sequences by analysis of expression structures. That adds more high level semantics to the user.
- In the same way: “class or equivalents” or “class of disjoints” elements can be defined on top of basic SHIQ axioms. When writing/reading to DAML+OIL corresponded transformations are performed.

Easiness of integration of ontological model / ontological edition user interface is crucial when ontologies are used within some application domains. There is an evident help from modern frameworks such as Eclipse and from component technologies. This is important, when research projects with sharp time frames are conducted.

By developing DL-workbench we have achieved all the requirements that were described at the beginning of this paper. Other important issues such as versioning support, openness of internal data formats and many others were considered during development. However, due to the limited size of the paper, we shall omit discussion of these topics. A reader can find some additional information on the web site of DL-workbench. Our current research for software integration is based on DL-workbench. We use described concepts for creation of extensions of DL-workbench that facilitate our experiments with integration of numerical solvers and creation of “good enough” ontologies verified by reasoner.

8 Other Editors and APIs

Many ideas of user interface were inspired by OilEd [6] ontological editor. OilEd is the first editor that implements most of the features of SHIQ description logic, reasoner connection and expression edition. That was an ontological editor we used before creating DL-workbench. However, despite of all its benefits, some elements of user interface, such as choice among “subClassOf” and “sameClassAs”, semantic of some axioms and some others elements are not always clear for the end user. We tried to resolve these issues in our tool. OilEd is an open source project, but OilEd API seemed to us difficult to be integrated with other tools. Presence of meta-model level within the DL-workbench gives more flexibility and ease of use together with other tools.

KAON API and a set of related KAON tools [14] define a distributed ontology manipulation infrastructure that is based on client-server architecture and provides many useful features. KAON has a hard coded API for its ontological formalism, that is mostly RDFS based and doesn't support extended semantics of equations nor very expressive description logics (such as SHIQ). We found the ontological editor OI-modeler of KAON to be difficult for the end user. DL-workbench doesn't support directly distributed interfaces or client-server architecture. Within our research (extension of DL-workbench), the meta-model module runs on each machine and exchange serialized representations of meta-model instances.

Protege [18] ontological editor has a convenient plug-ins API. Many useful plug-ins are written for the extension of Protege. However it's not so easy and natural to extend Protege with other ontological formalism and moreover we found it quite difficult to integrate Protege into other environment. Protege has a plug-in for ontology merging/alignment. Such functionality is planned for DL-workbench.

There exist many other ontological editors. Due to the absence of space in the article we can't make overview of many tools. [20] is a good survey of these tools.

We had an alternative choice to use OMG MOF repository specification [3] (which is used in many UML-related tools) for implementation of models. However we found current MOF implementations to be heavy for an ontological editor. Moreover, the use of DL notations for definition of meta-model elements makes it much clearer.

9 Conclusions and Future Plans

We have presented DL-workbench, both an Eclipse-based ontological editor for SHIQ logic and a meta-model based platform for manipulation of ontologies in conjunction with other tools. We've shown the benefits to use meta-model for creating of ontology-based products especially when working within specific application domains.

Today the DL-workbench is a research prototype and it lacks the stability that is needed for industrial development of ontologies. It lacks the functionality of merging of ontologies and more extensive support of reasoners features. All of that is planned to be corrected in the nearest future.

We use DL-workbench for development of our domain specific extensions and integration with other tools. That assures the constant evolution and support of the DL-workbench. In the future we plan to introduce a transaction mechanism with undo-redo operations, merging/alignment of ontologies, graphical representation of ontological information and make many other improvements.

Acknowledgments

This work is supported by Open Cascade S.A. We want to thank many people from the OMG business rule working group for useful mailing list discussions, special thank to Ed Barkmayer from NIST.

References

1. F. Baader et all, "*The Description Logic Handbook: theory, implementation and applications*", Cambridge University Press, 2003 ISBN 0-521-78176-0
2. The SALOME project, Online: http://www.opencascade.org/salome
3. OMG MOF, Online: http://www.omg.org/technology/documents/formal/mof.htm

4. S. Bechhofer, "The DIG description logic interface: DIG/1.0", 2002, Online: http://www.fh-wedel.de/~mo/racer/interface1.0.pdf
5. DL-workbench project web site. Online: http://www.opencascade.org/dl-workbench
6. S. Bechhofer et all, "*OilEd: a Reason-able Ontology Editor for the Semantic Web*", Springfied-Verlag, LNCS, 2001
7. I. Horrocks, U. Sattler, and S. Tobies. "Reasoning with individuals for the description logic *SHIQ"*. LNAI number 1831 pp. 482-496. Springer-Verlag, 2000
8. DAML+OIL language, Online: http://www.w3.org/TR/daml+oil-reference
9. OWL language, Online: http://www.w3.org/TR/owl-absyn
10. IBM Eclipse 2.1 platform, project page, Online: http://www.eclipse.org
11. Racer reasoner. http://www.fh-wedel.de/~mo/racer
12. FaCT reasoner, http://www.cs.man.ac.uk/~horrocks/FaCT
13. Java Swing UI widget library, Online:
14. KAON API, Online: http://km.aifb.uni-karlsruhe.de/kaon/Members/rvo/kaon_api
15. M. Kifer, G. Lausen, "*F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme*", 1990. Online: http://citeseer.nj.nec.com/kifer90flogic.html
16. Xerces XML parser, Online: http://xml.apache.org/#xerces
17. Jena DAML and RDF parser, Online: http://www.hpl.hp.com/semweb/index.html
18. Protege environment, Online: http://protege.standord.edu
19. M. Kazakov, H. Abdulrab, E. Babkin, "*Intelligent integration of distributed components: Ontology Fusion approach*", In proceedings of CIMCA 2003 conference, 2003, ISBN 1-740-88069-2
20. M. Denny, "Table 1. Ontology editor survey results", 2002, Online: http://www.xml.com/2002/11/06/Ontology_Editor_Survey.html

Context Modeling for Software Design

Tobias Berka

Salzburg Research, Jakob-Haringer Str. 5/III, 5020 Salzburg, Austria
tberka@salzburgresearch.at

Abstract. In this paper we illustrate the usage of a new formal model for describing systems and present a software design example. The presented approach is characterized by the possibility of describing a system's functionality very coarsely grained but formalized, allowing it to be used during early design phases while still being able to draw conclusions and provide support. The emphasis is on assisting developers, who utilize one or more methods from the field of artificial intelligence (AI) in complex systems. The synergetic effects and possible spin-offs can be quite numerous, and our proposed approach attempts to hide the complexity from the developer wherever useful, leading to a smoother and more efficient integration of multiple AI methods in a software systems.

1 Introduction

In software design and formal system specifications, with the multitude of well established methods and models one might hastily conclude there would be little need for new modeling techniques. But today's software systems are changing, which can be observed most clearly in the domain of web applications (see [6] on web engineering), which is clearly a candidate field for recent, more flexible and "agile" software development techniques (see [13] or [17] on agile methods). Today, various shades of *context* are used in different communities. Ubiquitous computing has created its own notion of context, adaptive hypermedia leads to user-adaptive systems (see [5] on adaptive hypermedia) and the semantic web (see [11]) can also be understood to provide knowledge adaptivity. As retrieval strategies become more widely available and thoroughly researched (see e.g. [7] or [10] as examples for such systems) these notions of adaptivity also lead to different notions of context.

We strive for adaptive systems, by using retrieval techniques which have been available for some time, but also computationally too expensive until recently. Computer based indexing systems have been used in libraries and public agencies for more than a decade, but now the web system community adapts these methods and integrates them into information systems.

This has implications in several aspects of web based information systems.

- It has a great impact on the authoring of content in such systems, where the user navigation can no longer be directly defined by the author, but new methods other than (or complementary to) hard-coding links directly are being devised.

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 34–45, 2004.

- The advanced functionality leads to new hybrid query strategies and types, and the effort of making all these query strategies available in a user interface in a convenient and comprehensive way is very high.
- The number of different information extraction processes which use very different information sources in such a system can be very high. We will consider the various extraction processes to be *queries to the system.*

This paper is concerned with the implications the development sketched above has for the *developers* of such systems, or "B People", according to D. Engelbart's definition (see [18]). Even if very few of these methods, which are well known in the artificial intelligence community, but rather new to web based systems, are used in one system, this can yield a *very* high number of operations on raw data and intermediate results. Some of these can give a system very advanced functionality at a very low development *price*, both in terms of development and computational time. We will propose a method, which can aid developers in managing the complexity of such systems, by reducing the complexity of managing the dependencies and synergetic effects in such systems.

1.1 The Approach

As we want to aid developers in a very early phase of software design, especially during the requirement analysis (see [8] on requirement engineering), we need a system which can draw conclusions based on very little and rather vague data. Traditional modeling conceptions using data-types and object-oriented specifications are too detailed to be used at such an early stage, and at the same time lack information which can be given very early during the design.

Our approach attempts to give insights into a *possible* system functionality, based on very little input. An implementation of the modeling support we propose would provide an interactive process of system functionality refinement, in which a formal model is used to provide recommendations to the developer. This means that as the system is described, the current specification can be analyzed and possible extensions extracted. The developer can then choose, which options are useful and valid, add these to the specification and refine and extend the system as needed.

In this paper, we will illustrate how our formal model can be used to describe systems, and what it can contribute to the process of software design by *deducing functionality* in an interactive process. Our basic unit of description will be that of a *context type.*

Definition 1. *An* abstract context type *is a token for all the dependencies from objects of one class to another, given a certain method for establishing this* context. *It is defined over* (*i*) *the method used to create or compute it,* (*ii*) *the class of objects it describes and* (*iii*) *the target object class of the dependencies.*

$$ContextType = (Method^{(i)}, ObjectClass^{(ii)}, TargetObjectClass^{(iii)}). \quad (1)$$

We can leave the nature of a method undefined beyond the ability to uniquely identify every method.

This notion of abstract context types allows us to specify functionality very easily, but in a formal manner. In the scope of this paper, we will not make any advanced assumptions about methods, as this would go too deep into the formal model.

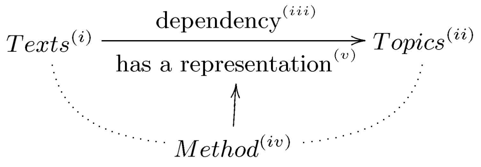


Fig. 1. Context Type Example

The process of specifying a system functionality as a context type is depicted in Figure 1. We begin by stating (i) the source object class and (*ii*) the target object class of a dependency (*iii*). A dependency used by a software system has a representation and can be *established* or computed by a method (*iv*). Using the method we can compute the representation of the dependency (*v*), and make use of the semantics and information of the dependency. But before we consider possible representations, we can make some additional assumptions about the our abstract context types.

1.2 Limitations

The basic machinery of our methodology can be seen as a system for inferring functionality and types which has a very “natural” limitation and which *cannot* be solved.

This underlying process uses a formal model based on sets, functions and rules. If we use the model to represent a system, these rules allow us to express the considerations on possible system extensions, which are of course subjected to the basic design principles in AI. This means that even though the deductions may be mathematically sound according to the model, and may be predicted, corrected, completed or reproduced by an underlying algorithm, there is no *mathematical* way of knowing the correctness of the “real world *interpretation*”.

If we study this process while considering the classical problem of frames (as in [16]), we can view the system design support as we propose it as an attempt to automatically deduce new frames based on the description of the initial frames designed by the developer. This cannot solve the basic “frames vs. reality” problem the field of artificial intelligence faces, but evolves around the idea of simplifying the task of designing modern systems with AI methods.

1.3 Related Work

The development of formal system models has a tradition going back to Alan Turing, and is thus a very old discipline in computer sciences. In this field, we have the classical background of finite state automata (see e.g. [9] or [23] on automata theory), and more flexible means of describing machine states (see e.g. [19] on petri nets). In these models, transitions between states describe operations, and graph representations and sets of operators as the mathematical basis for modeling.

Other formal models for system specification use logical predicates as their main mathematical notation, allowing much richer vocabularies and thus system descriptions (see [2] or [4] on logic-based approaches). Other approaches use algebraic formulations of semantics and types (see [12] for algebraic modeling), or richer mathematical models such as coalgebras (see [21] on coalgebras and systems) and category theory (see [1] and [20] or [24] on machines and category theory). We have adopted the latter as a primary means of mathematical modeling.

Our model (as in [3]) differs from these in the way, in which systems are represented. Instead of describing *individual states* of a system, we specify dependencies and the kind of information which a given dependency constitutes. In this paper, we will attempt to give an example of how the basic modeling concept can be used to model a system without an in-depth discussion of the formal model, and what benefits a developer can gain from using this approach.

2 System Modeling with Context

Our approach to system modeling evolves around the notion of context. The most prominent example of context as a modeling principle in computer systems is the *user access context* in m-Commerce and ubiquitious computing[1] (see [22] on m-Commerce). A softer and more widely known notion of context is the linguistic cotext - i.e. documents, and their *surrounding* documents (see [15] on computational applications of cotext).

There is a great variety of different forms of context in humanities, as well as algorithmic and implementational forms of context (see [14] for formalized context in predicate logic). The notion of linguistic cotext clearly originates in humanities, but can have an immediate application in web based information systems. Linking documents based on cotext has a very intuitive "feel" for a web system designer, even though the technical implementation can be quite complex.

In our approach, we use a very wide notion of context. As an example, let us consider the representation of very common notions in AI supported information systems, starting with very basic considerations.

- To web system designers, a context between users can be any form of user similarity.
- Linguistic cotext can be seen as a context between documents, giving a designer a method for document similarity.
- If we now use implicit vote collection to estimate the interest of a user in documents, we can interpret this as a context between users and texts.

[1] Or simply "context" in these communities.

If we now consider a web system using linguistic cotext, implicit vote collection and user similarity together, we already have a quite complex system. So with as little as three different forms of context we have a system which allows many different forms of query strategies and system interactions. We can formalize these three types of context as below:

$$t_1 = (cotext, texts, texts), \tag{2}$$

$$t_2 = (user\,similarity, users, users), \tag{3}$$

$$t_3 = (implicit\,vote\,collection, users, texts). \tag{4}$$

This way of expressing a system shifts the focus from static data-type representations to the interactions and dependencies in a system. For our purposes, a context *type* is defined abstract, merely as a token for the underlying dependencies. This allows us to describe a functionality very roughly, but very early during system analysis.

2.1 Abstract Context Groups and Connectivity

As we are dealing with an abstract notion, we have left the underlying *data* structure of an abstract context type open. But it is also clear, that the data type of a context type depends on the method used to compute it. As we follow a policy of refinement, we will ignore the types at this moment, but this means that we cannot make assumptions about the method. As we wish to analyze a system specification at this early phase, we introduce a generalization from the abstract context types to abstract context *groups*.

A context group is derived from one or more context types by dropping the method. This means that a context group is simply a pair of object classes, or a general template for *any* kind of dependency between these object classes, making a context group a token for several context types.

Definition 2. *An* abstract context group *is a token for one or many abstract context types. It is defined over the two object classes of the underlying context types.*

$$ContextGroup = (ObjectClass, TargetObjectClass). \tag{5}$$

The idea of a context group representing dependencies leads to a first rule.

Definition 3. *We introduce the context connectivity rule* $\circ$*, which is defined as a binary relation over the set of context groups. Two context groups are considered to be* composition candidates*, if (i) one context type's first object class matches another context type's second object class, and (ii) every context group can be composited with itself.*

$$\circ \subseteq ContextGroups \times ContextGroups \tag{6}$$

$$(A, B) \circ (C, D) \Leftrightarrow B = C^{(i)} \vee (A = C \wedge B = D)^{(ii)}.$$

This rule is one of several basic principles of the formal model, spanning across all representations of untyped views of context. In order to apply the connectivity between context groups to context types, we simply define that it holds for all pairs of context types, which have group connectivity if we drop the method.

Again considering software systems, we can use our new connectivity rule to analyze possible extensions of our system, without a single type specification.

2.2 Typing Abstract Context Types

If we want to composite context types our connectivity rule alone does not suffice. We need some means of specifying the key features of the data types involved. To model our representation of data types, called type concepts, we have two initial basic properties to consider: basic type and structure. The basic types are a rough taxonomy of implementation dependent primitives. The structure types evolve around different mathematical models and applications. Our allowed structures include **single values, vectors, matrices, lists** and **assignments**.

Apart from the assignments, these are fairly common notions. An assignment is simply a mapping from one object class to another. It can contain any means of translation between object classes. The concise nature of this translation must not be given at this stage, but it can be implemented after requirement analysis.

Table 1. Types and Simple Methods - Denotation

Symbol	Function Type	Example Function
d_R	single value distance	$d_R(a,b) = \|a-b\|$
m_R	combined value metric	$m_R(a,b) = \frac{a+b}{2}$
d_{R^n}	vector similarity	$d_{R^n}(a,b) = \boldsymbol{cos}(a,b)$
$\times$	scalar multiplication	$\times(v,k) = kv$
$\circ$	function composition	$(g \circ f)(x) = g(f(x))$
CSS	context similarity substitution	-
t	object class translation	$f: A \rightarrow B$ $a \mapsto b$

By considering type and structure, we can already make some basic assumptions. Vectors of equal primitive type and dimensionality *may* be candidates for vector similarity, values *might* also be comparable. A combination of a vector and a value may permit a scalar multiplication. We also define several other permitted operations, such as vector similarity for vectors with matching type, dimensionality and index semantics, combining single value metrics and other common AI methods (see Table 1).

There are various indications possible based on type and structure, but more well founded decision must be based on some specification of semantics. In our approach, we specify semantics over object classes. If we add a specification of

Table 2. System Specification Tasks

Interaction	*Output*
Specify Object Classes	Object Classes
Specify Dependencies	Context Groups
Specify Functionality	Context Types
Type Context Types	Type Concepts
Show Deductions and Confirm Changes	System Specification
Visualization and Browsing	Developer Support

semantics to a vector, we merely state what *object class* the indices are about, or derived from. If we consider a term frequency vector, the indices are about *terms*. This means that we would specify the index semantics over an object class *terms*. We now have a small but expressive vocabulary for expressing system functionality and drawing conclusions. Table 2 depicts the operations involved in the system specification process.

3 Modeling an e-Tourism Recommender System

We will now present an example of the process of modeling and designing a system using our context modeling approach in the domain of e-Tourism recommender systems.

3.1 The Initial Model

Our system focuses on the *softer* decisions in the domain, and neglects decisions, which can easily be done in an initial database query, such as position and time. We will first specify our object classes.

Users - Users are persons looking for travel recommendations.
Themes - Themes are main interest fields in tourism, such as sports, history, arts, music, relaxation or others.
Destinations - Destinations are static objects of interest, such as hotels, camping places or resorts. These usually have a lodging function.
Events - Events are dynamic objects of interest, such as concerts, markets, exhibitions or interesting sites without lodging function.

We now specify our initial context types. We have a user's interest in our themes in feature vector representation:

$$\begin{aligned} |User\,Interest\,Profile| &:: [Users] \rightarrow [Themes] \\ AppliedVector &:: [Themes], \end{aligned} \tag{7}$$

where $AppliedVector :: [Themes]$ denotes a numerical vector, where every component describes the relevance of a theme for the user.

We also have an assignment, which allows us to specify to which destination an event *belongs to*, which is usually decided over distance:

$$\begin{aligned} |Tourism\,Information| &:: [Destinations] \rightarrow [Events] \\ Assignment &:: [Destinations] \rightarrow [Events]. \end{aligned} \tag{8}$$

And we have a description of an event's theme in feature vector representation:

$$\begin{aligned} |Event\,Description| &:: [Events] \rightarrow [Themes] \\ AppliedVector &:: [Themes]. \end{aligned} \tag{9}$$

3.2 Deductions

Using just plain vector similarity, we can derive several context types.

A new context type for user similarity can be deduced using vector similarity on the context type specified in Statement 7.

$$\begin{aligned} |User\,Similarity\,over\,Interest\,Profile| &:: [Users] \rightarrow [Users] \\ AppliedValue &:: [Themes]. \end{aligned} \tag{10}$$

We can use vector similarity between our “user profile” (see Statement 7) and our event description from Statement 9 to deduce:

$$\begin{aligned} |User\,Event\,Rating| &:: [Users] \rightarrow [Events] \\ AppliedValue &:: [Themes]. \end{aligned} \tag{11}$$

But we can also deduce a similarity between events, again using vector similarity on event description (see Statement 9).

$$\begin{aligned} |Event\,Similarity| &:: [Events] \rightarrow [Events] \\ AppliedValue &:: [Themes]. \end{aligned} \tag{12}$$

As all events are known to the system, we can build a different vector representation for a user's interest, by directly specifying the rating of every event for a given user, and store it component-wise in a vector. This leads to a new context type, constructed from Statement 7:

$$\begin{aligned} |User\,Event\,Interest| &:: [Users] \rightarrow [Events] \\ AppliedVector &:: [Themes]. \end{aligned} \tag{13}$$

Going back to vector similarity, we can now deduce an expanded user similarity over events from our new interest representation as in Statement 13, which differs from our basic user similarity as it is computed over expanded vectors.

$$\begin{aligned} |User\,Similarity\,over\,Events| &:: [Users] \rightarrow [Users] \\ AppliedValue &:: [Themes]. \end{aligned} \tag{14}$$

Another possible operation is to use two assignments for translation, and the deduce a new context type by substituting a context type, which is compatible to the translated object classes. In our e-Tourism recommender system, we can deduce a new type:

$$\begin{aligned} |Destination\,Theme\,Description| &:: [Destinations] \rightarrow [Themes] \\ AppliedVector &:: [Themes], \end{aligned} \tag{15}$$

using this context similarity substitution, based on the translation from destinations to events in Statement 8 and the event to theme context type expressed in Statement 9.

Using scalar multiplication between a user similarity value and a user interest vector (Statements 7 and 10), we can build user interest expansions:

$$\begin{aligned} |User\,Interest\,Expansion\,over\,Users| &:: [Users] \rightarrow [Themes] \\ AppliedVector &:: [Themes], \end{aligned} \tag{16}$$

and another expansionover events, based on Statements 9 and 11.

$$\begin{aligned} |User\,Interest\,Expansion\,over\,Events| &:: [Users] \rightarrow [Themes] \\ AppliedVector &:: [Themes]. \end{aligned} \tag{17}$$

Again considering vector similarity on our previous deductions new context types, we can deduce a similarity for events based on the context type in Statement 15:

$$\begin{aligned} |Destination\,Similarity\,over\,Events| &:: [Destinations] \rightarrow [Destinations] \\ AppliedValue &:: [Themes], \end{aligned} \tag{18}$$

and now a destination recommendation rating for users based on the context type in Statements 15 and 17:

$$\begin{aligned} |User\,Destination\,Rating| &:: [Users] \rightarrow [Destinations] \\ AppliedValue &:: [Themes]. \end{aligned} \tag{19}$$

We can also deduce an expanded event rating scheme based on Statements 9 and 16:

$$\begin{aligned} |User\,Event\,Rating\,over\,Similarity\,Users| &:: [Users] \rightarrow [Events] \\ AppliedValue &:: [Themes], \end{aligned} \tag{20}$$

and craft composite ratings from values, such as a combined user similarity:

$$\begin{aligned} |Combined\,User\;\;Similarity| &:: [Users] \rightarrow [Users] \\ AppliedValue &:: [Themes], \end{aligned} \tag{21}$$

based on value composition on the context types in Statements 10 and 14.

3.3 System Queries

In this paper, a query is not to be understood as query in a web application, but more general as an "request for information" to the entire system, using very different information sources. For example, a query can consist of a database query, a deduction step in an expert system *and* a prediction of a probabilistic classifier - if the composition has been deduced and accepted by a developer. Based on our formal model, we can deduce the *computability* of the following queries in our system:

- Destination Recommendations - Context Type in Statement 19.
- Expansive Query for Similar Users - Context Type in Statement 21.
- Quick Query for Similar Users - Context Type in Statement 10.
- Event Recommendations - Context Type in Statement 11.

4 Summary and Conclusions

We have used a very coarse-granular and short initial specification of our system functionality to deduce a number of possible queries. Summarizing our above deductions, we can recapture that our initial system specification,

$$\begin{aligned} |User\,Interest\,Profile| &:: [Users] \rightarrow [Themes] \qquad (22)\\ AppliedVector &:: [Themes],\\ |Tourism\,Information| &:: [Destinations] \rightarrow [Events] \qquad (23)\\ Assignment &:: [Destinations] \rightarrow [Events],\\ |Event\,Description| &:: [Events] \rightarrow [Themes] \qquad (24)\\ AppliedVector &:: [Themes], \end{aligned}$$

led to several interesting possible system queries.

These deductions should be computed in an interactive process, which requires some amount of human interaction after every inferencing step, due to the high number of possible recombinations. Automation is both a key requirement and advantage of our approach.

In this paper, we have presented a modeling approach for software systems. Contrary to existing approaches, this approach allows developers to operate with very little information, and in a very vague manner. This, combined with the ability to formally deduce new functionality, makes this approach interesting in early software development phases. Especially during requirement analysis, this modeling approach can be very helpful, as the semi-automated system functionality deduction can work on very *natural* information.

Our next steps will be to develop ways to integrate process flow descriptions to our model, and integrate support for available description standards, such as basic class definition skeleton code for a few programming languages, and study in which ways we can tie in to existing database standards (SQL and OODB languages).

Acknowledgements

This work has been supported in part by the EU under the Information Society Technologies RTD project “VICODI” (IST-2001-37534).

References

1. M. A. Abib and E. G. Manes. Machines in a category. *Journal of Pure and Applied Algebra*, (19):9–20, 1980.
2. J. R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, Cambridge, UK, 1996.
3. T. Berka. A categorical context model - technical report, available at http://www.salzburgresearch.at, 2003.
4. B. Carpenter. *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, UK, 1992.
5. P. De Bra, P. Brusilovsky, and R. Conejo, editors. *Adaptive Hypermedia and Adaptive Web-based Systems, AH2002*. Springer Verlag, 2002. Lecture Notes in Computer Science.
6. Y. Deshpande, S. Murugesan, A. Ginige, S. Hansen, D. Schwabe, M. Gaedke, and B. White. Web engineering. *Journal of Web Engineering*, 1(1):3–17, 2002.
7. J. Domingue and E. Motta. A knowledge-based news server supporting ontology-driven story enrichment and knowledge retrieval. In *Submitted to the 11th European Workshop on Knowledge Acquistion, Modelling, and Management (EKAW 99)*, 1999.
8. P. Grünbacher, A. Egyed, and N. Medvidovic. Dimensions of concerns in requirements negotiation and architecture modeling. In Tarr P., Harrison W., Finkelstein A., Nuseibeh B., and Perry D., editors, *Multi-Dimensional Separation of Concerns, Workshop at 22nd International Conference on Software Engineering, Ireland*, 2000.
9. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Adison-Wesley Publishing Company, Massachusets, USA, 1979.
10. Y. Kalfoglou, J. Domingue, E. Motta, M. Vargas-Vera, and S. B. Shum. myplanet: An ontology-driven web-based personalized news service. In *Proceedings of the IJCAI'01 Workshop on Ontologies and Information Sharing*, Seattle, WA, USA, 2001.
11. T. Berners Lee, O. Lassila, and R. R. Swick. The semantic web. *Scientific American*, 5, 2001.
12. D. J. Lehmann and M. B. Smyth. Algebraic specification of data types: A synthetic approach. *Mathematical Systems Theory*, 14:97–139, 1981.
13. Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.
14. J. McCarthy and S. Buvac. Formalizing context (expanded notes). In *Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language*, pages 99–135. American Association for Artificial Intelligence, 1997.
15. A. Mehler. Text mining with the help of cohesion trees. In *Classification, Automation, and New Media: Proceedings of the 24th Annual Conference of the Gesellschaft für Klassifikation*, pages 199–206, Passau, Germany, March 2000. Springer.

16. M. Minsky. A framework for representing knowledge. In P. Winston, editor, *The Psychology of Computer Vision*, pages 211–280. McGraw-Hill, New York, 1975.
17. James Newkirk and Robert Martin. *Extreme Programming in Practice (The XP Series)*. Addison Wesley, 2001.
18. P. J. Nürnberg, editor. *MIS 02 – Proceedings of the first Symposium on Meta Informatics*. Springer Verlag, 2002. Lecture Notes in Computer Science (LNCS) No. 2641.
19. J. L. Peterson. *Petri Nets: Theory and the Modeling of Systems*. Prentice Hall Inc.
20. B. C. Pierce. *Basic Category Theory for Computer Scientists*. The MIT Press Cambridge, London, England, 1991.
21. J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.
22. N. M. Sadeh. *m-Commerce: Technologies, Services and Business Models*. Wiley, 2002.
23. Editors C. E. Shannon and J. McCarthy. *Automata Studies*. Princeton University Press, 1956.
24. J. van Oosten. *Basic Category Theory*. BRICS Lecture Series LS-95-1, University of Aarhus, Denmark, January 1995.

On the Foundations of Computing Science

Ulisses Ferreira

Abstract. In the present work, it is observed and demonstrated that the foundations of computing science, and even of science and logics, include forms of inference that are not regarded as valid, in neither logical nor scientific way. The present paper also shows two paradoxes of logics and scientific methods. Taking into consideration a certain rigor, the present paper argues that computing science is not mathematical logics, and that philosophy, psychology and other human sciences are in the foundations of that science.

1 Introduction – The Conceptual Diagram

This section and the following ones introduce one classification that can be used as a tool. An example of connection between this tool and the sample paradoxes that I will present below is the following: there exist two large classes of concepts, analytic and synthetic. Roughly, the former is the concept of *smallness* while the latter is the concept of *greatness*. Thus, the former can be seen as smaller than the latter and, hence, if in some observation a subject from the latter is seen as belonging to the former, there can be some contradiction, which can lead to a possible refutation.

As one of the possible conclusions from my PhD thesis, I identify some of the usual concepts of computing science and place them in only four classes. Therefore, here I attempt to justify the present classification but knowing that it is a transcendental matter impossible to be either proven or refuted or both.

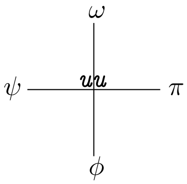
Most attempts to classify real world concepts necessarily lack precision, although attempts are often helpful. Because such classifications are empirical but takes a life time, here I present a classification from my standpoint, which illustrates the orthogonality of some paradigms and issues, skills, methods and approaches in the theoretical foundations. I illustrate how I identify the analytical and synthetic sides of computing science. Due to its empirical characteristic, our four-kind diagrams can be seen as simply a diagrammatic form of representation by some holistic view in computing science. Although fuzzy systems have been regarded as logics in their own right, I place these two notions as opponents in those diagrams, and this is because logics (which one can see as an analytical subject) is traditionally seen as the subject of the valid reasoning, hence, has some general meaning for all individuals. In contrast, fuzziness (which I see as a synthetic subject) typically requires individual truth threshold for each hypothesis, while fuzziness suggests, for every hypothesis, different truth threshold for different individuals. Furthermore, for I am placing logics and logic programming in the same analytical side in the referred to diagrams, and I am stating that

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 46–65, 2004.

any hypothesis that involves a synthetic subject cannot be generally proven, i.e. with a universally-quantified claim, the thesis of the validity of these diagrams, including my synthetic and empirical classification, is not provable or refutable.

2 Synthesis in Programming

Most work in computing science and AI can be very briefly represented by the following diagram:



One of these hypotheses that I am presenting here is that the programming paradigms which I represent here in the Greek letters that were carefully chosen, π, ω, ϕ, ψ, for example, (functional-logic programming, equations, constraint programming, consistency can be placed in π), (imperative features including side-effects, states, communication between machines, interaction, small mobility can be placed in ω), (internet programming, strong mobility can be placed in ϕ), and (uncertainty, fuzzy systems, inconstancy can be placed in ψ), are orthogonal ways of seeing the world from the programming standpoint. Given this, Plain[11] programming language takes into consideration this orthogonality, that is, different points of view. Moreover, Plain provides ***uu*** (in the center of the diagram, above), which is a constant for representing the non-information. The original idea of ***uu*** in logics is back dated to Łukasiewicz, but I extended the same idea to the context of programming languages[12].

Programming is a relatively complex task and, because of this, every programming language has features relating to all kinds. However, one can analyze and perceive predominance to one specific kind in most programming languages. The language Smalltalk, for example, can be associated to the ω kind; while Haskell could be the best example of the π kind. Concerning the ϕ kind, the mobile-code languages for global computers nowadays are still very emphatic on code mobility. More specifically, they have not solved all practical problems with respect to security, which itself is only one problem. There are other issues, e.g. regarding differences of cultures, that have not been investigated. In fact, the ϕ kind is too recent from the programming perspective and one misses comparisons on scientific basis. Plain tries to achieve a suitable balance between these four kinds. So, in the next subsections, I shall discuss the four kinds and make comparisons between them.

2.1 Constructs and Concepts of Kind π

This kind of constructs is not easy to describe because it seems that all programming languages are in here. Formalism, discipline, precision and details are extremely important, and they are key motivations here. Most logic programming languages and functional programming languages are typically in this kind, and because strong typing goes together with discipline and methodology, these languages are not of the same point of view as untyped programming languages, which are in ω, for instance. Objects are concrete and well formed here. Although most of object-oriented programming languages are imperative, inheritance is another example of constructs of kind π. There are exceptions such as C++, which in turn has a strong influence from C. Here, sub-classing can be sub-typing in terms of OOP. The senses of order and pureness are important here. And because of the importance of order and clarity in programming, I regard the kind π as compulsory.

For this synthesis, because there has been much work on logic programming and functional programming, I have preferred not to concentrate on languages of this kind. Instead, in the following subsections, I use them only for comparison.

2.2 Constructs and Concepts of Kind ω

Here flexibility and expressiveness are keywords. Others are interaction, communication, states, imperative features, and efficiency of time, for instance.

It is interesting to note that, although C programs do not tend to be robust, C is among the most successful programming languages and its historical importance is undeniable. Java is one of its successors.

The most traditional languages used in knowledge-based systems are Lisp and Prolog, which are untyped programming languages, although based on functions and logics, respectively, which are in kind π. In both languages, *list* is a basic data structure that provides flexibility.

Although Plain is a strongly typed language, because its designer and other users require flexibility, Plain also permits heterogeneous lists in programs. The head of a list is accessed by using the name of the type as the head function call. It has been a nice experience to program with heterogeneous lists, and programmers should be pleasant to program. The following is an example interpreting [ ] as an empty list:

$$\begin{aligned} \textit{public list } reverse([\,]) &= [\,], \\ reverse(li) &= reverse(tail(li)) + head(li); \end{aligned}$$

In the above example, the *public* keyword states that the function $reverse$ can be used by another agent. The + operator here concatenates two lists. The *head* function gives a list with only the first element of its argument. Therefore, the above function reverses the order of the elements of a list regardless of its type.

For this kind of list, the programmer can infer types[20] or, alternatively, instead of the word *head*, use the type identifier to get its first element. In this way, at the point one writes the type identifier in the code, the type checking is made at runtime. In this way, that typical function definition becomes here as follows:

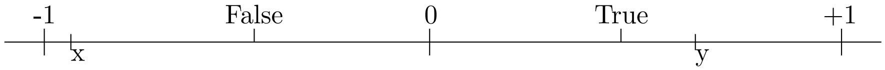
$$\begin{aligned} \mathit{public\ int\ add}([\,]) &= 0, \\ \mathit{add}(\mathit{li}) &= \mathit{add}(\mathit{tail}(\mathit{li})) + \mathit{int}(\mathit{li}); \end{aligned}$$

Thus, the above function computes the total value of the elements of a list. Such function definitions, together, illustrate that untyped lists and strong typing can coexist at the language level. A question is how to reconcile the flexibility of programming with the methodologies of pure languages, of kind π. Another question is whether type inference[30,36] is better or worse than otherwise with respect to programming methodology. For instance, it can be seen that type inference makes *programmers* infer types. Programmers write a piece of code only once and, later, will look at it many times. Therefore, programmers normally infer types in languages like ML. This can be seen as a contrast between kinds ω and π, for the former is predominantly based on assertions and *facts* (type declarations) whereas the latter is predominantly based on *deduction* (type inference). Therefore, this pair of different points of view, ω and π, often indicates the issue of facts (e.g. predicate declaration) and deduction under either open- or closed-world assumption. The present author has proposed the former in [13].

2.3 Constructs and Concepts of Kind ψ

Here, uncertainty, inconsistency and relativity are some of the keywords for this kind of constructs, ψ. We can include here any constructs for knowledge representation that have vagueness. One approach is to avoid being impersonal and exact, in the programming paradigm, because of the consistency of this perspective. If so, surprisingly, probability theories may not enter this class, in spite of their undeniable importance in science, in general. In terms of programming and knowledge representation, exact probabilities are not easy to be found in the real world, let alone being represented using some formal language. It is known that probabilities are based on assumptions concerning whether events are related to each other. But one underlying subjective issue is “when can we consider that two events are independent from each other?”. There seems to be no objective or precise answer for this question. In fact, depending on the philosophical point of view, I have totally different answers. For programming with probabilities, there is one problem of computational complexity[8] and another of representing the web of events, which is a difficult one. A similar web is represented using ad-hoc models, where the basic difference is that probability is traditionally mathematics while ad-hoc models are not so recognized. This means that probabilities require too precise values for representing imprecise knowledge about the real world.

Here, pictures, images, films, and diagrams can represent the imprecision and subjectivity of the real world in a more suitable way. My proposed programming model for uncertainty extends MYCIN to predict the evaluation of hypotheses by introducing two variables, the least and the greatest factors of certainty, respectively denoted as x and y in the following diagram. As a result, PLAIN provides an intermediary state, called uu, since we have two thresholds for $false$ and for $true$ as follows:



Since x never decreases and y never increases, reasoning is monotonic in this setting. However, if I force $x = y$ constantly in the above setting, it is as non-monotonic as almost all expert system shells with attached certainty factors (most of them are based on production rules). In this way I have two alternative options for all hypotheses. Monotonic reasoning suggests that the truth is only one. By providing two variables, computations also have the ability to predict, at least in some sense, and that is *essential*: the value $y - x$ represents the variables in the program that have not been explored and can contribute to prove or refute the hypothesis. The investigation of the veracity of these variables can be very costly. The system therefore anticipates the evaluation of the hypotheses in accordance with the representation, above. This is similar to *lazy evaluation* in the sense that *variables in the premise list are evaluated at most once.* In this model, inconsistency can be obtained in the form of $False > True \vee x > y$, if the language and its implementation allow this situation.

If one user wants an agent to represent him or her, he or she wants to personalize the behavior of his or her agent if the task is not simple. Due to the huge number of different certainty factors, the community needs suitable languages to program agents.

2.4 Constructs and Concepts of Kind ϕ

In ϕ, among other notions, as humans, we learn by induction and analogy, and also use metaphors to communicate. We humans need to make comparisons and need broad and abstract views. And since we have had a broad or general perspective, we are able to start solving hard problems in an easy way, and then to investigate them deeply, top down. Aims are necessary, but they should be set only after having such a view and before solving the problem in question.

We humans cannot investigate some complex subject deeply without an initial broad view. Goal-driven programming, i.e. where programmers set goals and the underlying system does the job, is motivated by ϕ. In this sense, Prolog and relational languages have this feature in ϕ. As another example, inductive logic programming is a combination between π and ϕ because, while logical and formal propositions are there, induction and learning by experience are in this class.

Because of its synthetic nature, this class of notions is difficult to implement on a computer, but I am advancing in this direction. Mobile agents, for instance, are applications whose motivations fit in this kind ϕ since mobile agents are flexible and free enough to gain experience abroad.

As regards "mobility", the difference between constructs of kind ω and ϕ is that, in the latter, differences of cultural and religious backgrounds are probably involved, while in the former, mobility essentially is related to changes of states, concurrence, unexpected effects, higher-order functions and hardware circuits: everything happens in the same machine in ω. While ω is driven by issues such as flexibility, communication and curiosity, here truth is a key motivation. In comparison to π, ϕ is not mainly concerned with syntax and details but instead broader concepts, such as paradigms, and also semantics.

In comparison to functional languages in π that adopt type inference, for instance, from the ϕ standpoint, I observe that type inference makes programmers infer types. To answer this kind of question there ought to be experimental and empirical research, not proofs. Plain is experimental since it was not conceived for commercial purposes. This is validated by twenty years of experience in programming and compilers. The Unix system and C are two successful examples of what one or two professionals can do, in contrast with PL/I, which in turn was designed by some greater number of people. In this way, it can be better to rely upon one's own experience than to make experimental research among non-experienced programmers, while the result of one's work may happen months or years later.

The fewer the number of programmers the more independence a designer has for trying different constructs. The right moment to release a language to others is relevant, and I am nearly at this point. Yet, by using Plain, the author has experienced important insights concerning languages and paradigms. One of the key ideas in Plain is its hybrid and well-balanced paradigm, i.e., it tries to combine well-balanced features from different points of view, while accepting that divergences among people and, hence, researchers are natural. Although Plain is a large language in comparison to functional languages, it is meant to be concise and relatively smaller, as it provides common constructs. Here rests the difference between a multiple and a hybrid paradigm, at least in comparison to a naïve approach. I have programmed and experimented with different kinds of constructs. The implementation should be sophisticated enough to hide complexity. On the other hand, a good hybrid paradigm is not hard to learn, since learning can be incremental, and makes programming much easier, since I have taken into consideration different kinds of motivations. Thus, if a person likes functional programming, it is easy to program in Plain, and if a person likes object-oriented programming, it should be equally easy to program in the same language. However, although the Plain philosophy is to be a hybrid language, the concept of "pure function", for instance, (or simply function. It is the opposite to imperative function) is present in the language, they are declarative, never making use of global objects, and this is guaranteed at compilation time. However, functions can be applied from any code of any paradigm. In this sense, Plain is different from PL/I where the key motivation was to bring together features for both engineering applications and commercial applications in *the same* paradigm. I believe[15] that, after some time, programmers naturally find that applications suggest the paradigm to use, and that they can benefit from a hybrid language after some time using a particular paradigm. However, at the present point it is an open issue that deserves future validation and perhaps even further work.

At the practical level, issues such as robustness and security are keywords in the mobile agents field of research. Even in programming, in society, laws have to be established, which stresses the relevance of philosophical issues in programming. There are other keywords, such as dynamic linking (which I did not adopt but may be necessary for efficiency), naming, and global-scope identifiers, i.e. "global" not in the old sense of global variables. The subjective aspects of such issues suggest new or open problems.

3 Synthesis in Knowledge Representation and Reasoning

For AI researchers, although the present classification is not complete (for instance, vision is important for AI and is a synthetic notion related to ψ up to some extent), the four-kind diagram can be seen from the knowledge/belief representation and reasoning/inference standpoint as follows:

- π: perception, precision, specialization, functional programming (traditionally LISP programs), logic programming (traditionally Prolog) and inheritance in class- or frame-based systems. Deductive rules, consistency, deductive reasoning and search. Analysis, complexity and efficiency. Closed-world assumption and negation as failure. Learning by deduction.
- ω: reasoning, in particular non-monotonic, natural negation, ambiguity and redundancy, knowledge, interaction, diversity, curiosity and learning by acquiring facts.
- ψ: feeling, fuzzy logics, uncertainty, partial information, incompleteness, subjectivity, inconsistency and belief.
- ϕ: intuition, perspective, monotonicity, absolute truth, speculation, axiomatization. Semantic web, inductive reasoning, generalization, analogy and metaphors. Objectivity as opposed to subjectivity. Open-world assumption, neural networks, inductive logic programming, broad view, synthesis and common sense. Learning by induction.

In terms of AI, this four classes can represent four types of intelligence. I observe that although the above concepts are interesting for AI, the classification is still the same as for programming languages and for computing science, which will be presented in the next section. While ω and π are analytical kinds, ϕ and ψ are synthetic ones. For instance, a country could well use many deductive rules of logic programming to decide whether a person may be regarded as a citizen of that country or not. However, deductive logics is not very appropriate for, in the airport, deciding whether a person from abroad may enter the home country or whether the person will go back to the place from where he or she came, because the number of rules from the real world is practically impossible to count. Therefore, as programs need synthetic thinking, programming languages should also provide synthetic tools, in particular for mobile agents. In [14], I presented constructs with uncertainty for filling in this gap. Continuing, while ω and ψ are closely related to internal judgments (by reasoning and feeling, respectively), π and ϕ work like input devices (by five-sense perception and intuition, respectively). This classification has strong influence from Carl Jung psychological types[22], but the observation with respect to computing science is almost entirely based on my own experience in both areas as well as my empirical observation during my PhD studies. Perception above, in the π kind, corresponds to a refinement of what Jung called *sensation.* According to him, the four psychological functions are: thinking, feeling, sensation and intuition. More generally, he called the first two *rational functions* while he called the last two *irrational functions.* These four functions are somewhat similar to what I called ω, ψ, π, ϕ, respectively. Philosophically, Jung's work itself had some influence from Immanuel Kant.

In terms of humanity, the essence of ψ is seen in the romanticism. An atheistic view of ψ is exposed in [34].

Regarding mobility, I can classify its scopes as (individual,ω), (social,π), (geographic,ϕ) and (universal,ψ). With respect to languages, natural and artificial, I can still observe differences among the main concepts: (vocabulary, ω), (grammar and syntax, π), (semantics, ϕ) and (pragmatics, ψ), for instance.

4 Synthesis in Foundations

Mobile agents and the Internet have brought new ideas to theoretical computing science in the last few years.

As an example, I recently had a conceptual insight over computation: "...at the moment that I conceive the idea of moving computation from one place to another, I also observe that a general notion of computation transcends pure mathematics and meets the physical world". This itself requires new, informal and philosophical discussions.

Regarding the two-axis diagram, there probably is the same correspondence in computing science at a very high level. Respectively, psychology (ψ), sciences related to the machine, engineering and physical characteristics in general (ω), mathematics (π) and philosophy (ϕ) form a four-leg table that can support computing science. Typical questions in science can also be placed under this classification e.g. *what*, *where*, *when* (factual and flat information in general, ω); *how* (π); *why* (ϕ); *for what* and *for whom* (ψ). *Who* can be either in ω or ϕ. These four legs are not sharp, too clear, exact or mathematical classification, e.g. in philosophy, the Platonist view can be placed in ϕ while the constructivistic view can be placed in π. On the other hand, in logics, theories of truth, interpretations and model theory can be placed in ϕ while proof theory can be placed in π. These four kinds are somewhat fuzzy[26], relative, incomplete and personal. However, the hybrid semantics of that four-kind diagram contrasts with solely fuzzy views of the universe, e.g. [26].

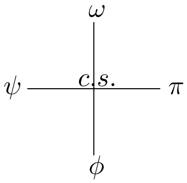
I have presented two different philosophical views in computing science, that can be summarized in the following way, depending on the adopted meaning for "executable code".

- Traditional view: computation has purely mathematical semantics. There are modal notions, such as mobile computation. Executable code is local to the machine, and a machine is only a physical notion. Correspondence between semantics[31]. Domain theory [1,3,19], denotational semantics [2,38,39]. Curry-Howard isomorphism [18], rewriting systems[4,25], category theory[27,28,41], functions, logics in computing science and so forth. Functional programming, input and output operations as monads.

- An alternative philosophical view: computation by machines is a unique physical notion. Executable code is mobile, and a machine can be an interpreter. The equivalence between operational and denotational semantics does not necessarily hold from this point of view. In particular, although denotational semantics is still useful, the operational semantics is the most suitable semantics to capture this view of computation if the notions of space and time are part of the semantics. In the real world, uncertainty- or probability-based computation is performed. Computation can be incomplete. Space-time logical calculi. Input and output operations as physical interaction (by side-effect) in the real world.

Notice that these views are not necessarily holistic, although I use the second view to illustrate "computation in the real world" in the present paper.

I can refer to computation in the latter philosophical view as not only computation carried out by machines but also *computation by humans*. For example, interaction for a machine is roughly equivalent to human five-sense perception. However, as will become clearer in this section, while there is some (rough) equivalence between human thinking and machine computation, whether one can reproduce or only simulate human computation in a machine is one more philosophical issue in the foundations of computing science. Continuing the discussion with a diagram:



There is the ϕ kind since we all also learn by induction and analogy, as well as we make use of metaphors to communicate, mainly when the concepts are abstract, and we have only words. Possibly, that is why prophets and visionaries often use metaphors in their speech. And because we humans make use of induction, analogies and also use metaphors in our speech[7], a number of good examples are both didactically relevant and essential scientific method, as computing science, in the broadest sense, is not totally mathematical. While in ω I talk about knowledge representation, in ψ, one regards with *belief* representation. Although the forms of representation can be the same, these two concepts are different. In ϕ, broad view and intuition are key words that normally lead to prediction[24].

In logics, while some of the main motivation in π would be proof and syntax, some of the main motivation in ϕ would be (perhaps informal) models and probably non-mathematical semantics. Since Gödel incompleteness theorem, it is known that proof (π), truth (ϕ) and incompleteness or inconsistency (ψ) are not equivalent notions in both the mathematical world and the real world. While, in π, *true* and *false* would be mere symbols that are manipulated according to some well-formed syntax, here in ϕ models would be closely connected to the real world. While in π a key motivation is to find differences between apparently similar objects, here in ϕ, a key motivation consists in finding similarities between objects apparently very different from each other. And while π can correspond to Aristotle and deductive reasoning, ϕ can correspond to Socrates and inductive reasoning[35]. Thus, we all need a sufficiently broad view of the world to make good analogies: the broader the view, the better the analogy. Metaphorically speaking, if I place elements of a set in order and I want to connect elements that share a property different from the chosen order, I have to see distant objects, perhaps at once, to make comparisons. Following this reasoning, the notion of perspective and, therefore, distances and mobility, can be obtained. At a somewhat refined level, induction is an orthogonal notion in the sense that it depends on experience and, hence, time.

While in π the related quantifier is existential, here in ϕ the related quantifier is universal. In terms of games, while π may summarize the skills played by Eloisa ($\exists$), ϕ can represent the skills played by Abelardo ($\forall$).

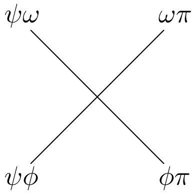
Many aspects of computing science have always had interesting philosophical components. As part of future work, one may want to define those bases explicitly, in a way that they can be identified by undergraduate students over the world. As well as philosophy, there are other human sciences which are important for the foundations of computing science.

With respect to the ψ kind, for being a transcendental notion, while π is concerned with granularity and what can be perceived and countable, ψ, like a set of water molecules, although a countable set, suggests what is uncountable and is beyond one's five senses. Like in modern physics, ψ suggests hypotheses and observable models, but not what can be directly perceived. Thus, belief is an important issue here. ψ, as I lexically suggest, is a place where psychology has something to contribute to computing science. By comparing and stating differences between humans and machines, we humans also learn the limits of what can be computed by machines, an approach complementary to complexity. As an example, if someone wants to build some application for identifying e-mail messages that are important or interesting, he or she has to have a very personal agent computation, probably based on some analogical measure, not only deductive systems such as natural deduction and Prolog, in which, for each consequent all pieces of its (conjunctive) premise must necessarily be tt. Thus, the notion of "importance" is not only personal but it also naturally requires some analogical interval.

Returning to the π-ψ axis, I can state that ψ suggests computation in the set of complex numbers, while the π kind, although $\pi = 3.141592...$ in arithmetics, is mainly concerned with the set of integers. As a synthesis, Gödel incompleteness theorem, at another level of abstraction, is a piece of work which involves five different notions that I have identified in the above conceptual diagram:

(π: proofs and consistency), (ϕ: truth and validity), (ψ: inconsistency), and finally (uu: incompleteness).

I can divide the same diagram in two diagonals as follows



forming two divisions, each of which one can be referred to as *side*. I could refer to ϕ and ψ as "synthetic side of the foundations of computing science" while ω and π would be "analytical side of the foundations of computing science". Therefore, this representation of those foundations of computing science would include the whole picture. The $\phi\pi$ side can be referred to as inductive and deductive reasonings, as well as knowledge/belief acquisition, whereas the $\psi\omega$ side can be referred to as knowledge/belief representation.

With respect to synthesis and analysis, which divide the diagram in two halves, $\psi\phi$ represents the purely subjective side of the foundations of computing science, while $\omega\pi$ represents the objective side of the same foundations. I simplify the language and refer to them as synthetic and analytical sides, respectively, having in mind that they are *not* independent from each other. The former is predominantly inductive while the latter is essentially deductive. In computing science, the analytical and concrete side has been well developed while the importance and even the presence of the synthetic side has been little perceived. This contrast is probably not simply due to the fact that deductions are very easy to implement in a machine. Indeed, while formal proofs are appropriate as a scientific method for the analytical side, *concerning the synthetic side, formal proofs of general claims do not work.* Frege[40] tried to prove that arithmetic was analytic[21]. As an example, tableau calculi are also called the method of *analytic tableaux.* From [5]: *Any calculus that starts with the formula to be proven, reducing it until some termination criterion is satisfied, is called* analytic*. In contrast, a* synthetic *calculus derives the formula to be proven from axioms.* In contrast, I am referring to *analysis* and *synthesis* as opposite mental processes. In fact, not only deductive logics, in its pure sense, is normally analytical at a higher level of abstraction, for Babbage himself called his work *analytical engine*, and so forth. These pieces of evidence are too informal and synthetic for being used as proof of the validity of this classification. In contrast with logics, there are branches of mathematics that seem to be synthetic.

Because of this contrast between synthesis and analysis, for some logicians, inference based on uncertainty might not be considered as part of logics, at least from the point of view of deduction and the universality of the classical logic. In the present work, I let them be opposites to form the same axis, as, in a sense, they complement each other because their natures are essentially inductive and deductive; synthetic and analytical; on open and closed word; subjective, and exact and with valid values for all; fuzzy and precise etc, respectively. Interestingly enough, logics may be seen as a branch of philosophy, which I classify as synthetic. It is true that nowadays, as there are many logics, to propose a logic one has to have a broader view than he or she would need to simply use the same logic.

I can also see the above diagram from the standpoint of the other diagonal and refer to $\phi\pi$ division as *perception* and *learning* while $\psi\omega$ division may be referred to as *reasoning* and *thought*. Perception here means not only by using five senses (programmers and logicians have to pay attention to forms and details) but also intuition (philosophers and researchers need to make use of their own insights and see the world abstractly, from a broad perspective).

An interesting explanation, extracted from [10], as regards intuitionistic logic: *"What distinguishes the intuitionists is the degree to which they claim the precedence of intuition over logical systems and the extent to which they feel their notions have been misunderstood by classically trained mathematicians and logicians"*. The idea of precedence of intuition over logical systems[32] is in accordance with the idea of trying to view the whole picture without details before starting concentrating on the latter, in a top-down fashion, for those who like software engineering. As known, intuitionism is a philosophical view[9].

As well as perception and intuition[29], in the above diagram, by completing the symmetry, thoughts are not only based on reason (to deduce hypothesis) but also based subjectively on feeling. Feeling is very personal. Even if the researchers decide not to implement such subjects as part of limitations of what computers can do, those subjects are still essential to the foundations of computing science. Nevertheless, it is easy to understand why this ψ kind has not been exploited in computing science, and an answer is that this is also a natural consequence that computers have become increasingly complex. Deep Blue has already beaten the Grandmaster Kasparov in chess, but computing scientists took some time until the machine was able to beat him, a challenge which could be regarded as relatively simple, besides its computational complexity.

So far, there has not been any implementation running in the computer that could be regarded as representing feeling, at least in a universal way. Nonetheless, the notion of agent was introduced to represent people in their transactions in the daily life. The diagonals of the diagram also represent the idea that, among many other skills, human beings learn by communication and facts, perception and deduction, intuition and induction, feeling and belief. The diagram is not complete with respect to this either, for instance, motivation and pain are outside the present classification.

The computing science and AI communities have been discussing the differences between humans and machines in terms of the meaning of thoughts. For instance, in [5], the same applies to discovering proofs and deduction. Even with a set of wffs, deduction might be difficult for a machine due to the possibility of combinatorial explosion. Here one has two philosophical views supporting the answers for the question about whether we humans are machines, or whether we humans are much more than this. Not only here but also as part of my PhD thesis dissertation, I have presented, as examples, some skills that form the synthetic side in the diagram. Although specific answers for such questions are outside the scope of the present piece of work, one of the main results from my observations is that, *if one wants to adopt the view which equates any person with any machine, first it is a good idea to study intuition, feeling, analogy, induction, belief and other subjects which are parts of the human beings, and thus, philosophy and other human sciences.*

Finally, technology introduces useful insights into theoretical computing science. The introduction of mobile agent technologies has led the community to want agents to be autonomous and flexible to represent users. For simple tasks, there is relatively little to add to what we humans already know. But users want to use complex application programs.

To truly represent people in our complex society, agents *have to* simulate subjective thought too, we humans want them to behave based on our personal tastes and personal opinions, for instance. Therefore, knowledge or belief representation can also be seen as a programming paradigm, not only as a subject in AI. The aim of introducing subjectivity in programming languages is difficult and ambitious. The author used to work on an expert system for diagnosis of heart diseases and was intrigued when observing that the users, experts, did not want to attach real numbers (in fact, floating-point numbers) to our rules as part of the certainty measure. Instead, I made use of words, carefully chosen to represent those real numbers. From that experience, the rôle of subjectivity in programming could be felt: "What do such subjective words mean?". There seems to be some common agreement which, in those cases, is more important and easier to understand in our every-day life than numbers. More recently, intelligent agents have been conceived to represent users and, for some reason, some of us also want them to be mobile. This scenario leads the present paper to conclude that other more subjective, less exact sciences are also essential in the foundations of computing science.

The intuitive part of this work is also based on a symmetry in pairs of four key concepts, namely, knowledge(ω)-intuition(ϕ), deduction(π)-belief(ψ). A difference between knowledge and belief is that belief is a kind of weak knowledge. In this sense, there exists some threshold between the two, and which determines what is knowledge and what is belief. Furthermore, this threshold is itself of course synthetic, and somewhat subjective. In short, a matter of belief. As an example, although it may be safe to state that most men acknowledge that Marilyn Monroe was beautiful, it is reasonable to think that beauty is subjective.

5 Sciences and Deductive Logics

Based on observation, we can classify concepts related to computing science, such as methods, forms of inference, mental abilities and subjects of research, dividing them in two classes, i.e. synthetic and analytical. This classification, for being original, differs, for example, from the classification of Immanuel Kant[23].

Here, there is no formal nor precise definition of *synthetic* and *analytical.* Roughly speaking, some analytical idea is motivated by exactness, whereas some synthetic thing is motivated by generality. In this way, this classification itself is synthetic. The analytical class is divided in two subclasses, namely, ω and π, whilst the synthetic class is divided in two other subclasses, namely, ϕ and ψ. Deductive logics is a key concept that belongs to the analytical class, whereas induction and analogy are two of the key concepts that belong to the synthetic class.

Logics and exact sciences, in turn, as it is known, are partially based on deductions as well as observable and deduced facts, while such facts can be used as examples, which in turn can be used as proofs for existential propositions. On the other hand, propositions based on a finite number of cases are traditionally regarded as less relevant. As an example, it is known that belief, induction and analogy have not been accepted as valid methods of sciences[33], in particular, mathematics and exact sciences.

Nonetheless, synthetic concepts are significant in computing science and, among them, there are those empirical concepts, together with belief, induction, analogy and so on. A few forms of inference, such as the ability to weigh up possibilities, can be deeply studied in computing science, while the ability to weigh up possibilities is not traditionally regarded as a mathematical method.

Here it can be observed that computing science not only profits directly from logics and mathematics but that that science has a direct connection with the real world, while it belongs to the same world. Furthermore, the typical place where computing science has many notions of the synthetic side is in the interaction with the real world, including the interaction with our senses at work, and in the *applications* of the analytical notions. An example of this is in section 5.1. I see computing science as the organized knowledge about the world. In fact, while the logical, scientific and mathematical kind π in the foundations of computing science has been well studied since Babbage, no significant contribution was done in *philosophy of computing science.* Today, this is a new side of the foundations of computing science.

In general, these synthetic aspects are subjects inside philosophy and human studies. Citing Immanuel Kant[23], “it is absolutely necessary that humanity believe in God”. Science and religion have been traditionally very separate from each other. Nonetheless, like religion, science seeks the truth, although its methods are constrained.

In this Kantunian sense, what is most useful to science is not only to acquire knowledge on whether Jesus of Nazareth really existed, or what he did, in small details etc. The concept of one or more entities morally superior to humans, and who have the properties of omnipotence, omnipresence and omniscience, certainly has relevant impact on sciences. As a clear example, if an individual is atheist and their proposal is very ambitious (for example, Sigmund Freud, for there is some evidence[6]), selfish or urges for power at any cost, he or she may manipulate results or forge input data in the absence of other scientists, or may steal someone else’s ideas or the credits of someone else’s work, as what he or she knows is that they are not going to be punished, whereas, typically, religious scientists do not *normally* do those for they know that, later, they would be somehow punished for the fault in question. Therefore, whether we humans believe or not, God's eyes play some relevant rôle in science.

In the short history of computing science, there has already been the public case of the suicide of Alan Turing, with the additional observation here that, as it is known, such an action can be seen as very anti-ethical, and the reason may be as follows: Given that all deaths, in particular suicides, should be known to the scientific community, if the whole humanity had been influenced by him and had done the same sort of thing, science would be extinct together with the humanity. Therefore, any suicide is an action against science, at least it should be seen as such, and this makes verifications of all the individual's work appropriate, if he or she had had international or historic reputation but made such a moral mistake, while scientists' personal lives should almost never be of great interest for the public. As well as his or her work, the scientist's ethic-religious system is what is important for others. In the case of Dr. Turing, he may be a hero for his work during the II world war, but his suicide was a public action somehow against the reputation of his own computing science.

Likewise, striking from behind by gossips, slanders and non-authorized (personal or professional) references are also very anti-ethical as, like a chemical or biological weapon, they irradiate in the academic and scientific community without giving any opportunity to the victim to defend himself or herself, at least in time to avoid losses, if the victim is not intelligent and wise enough, which is not the general situation. And because sciences seek the truth, other researchers are also victims of those lies in question. Therefore, given that the material of computing science is information, ethics is essential indeed, and should be taught as part of the formal course.

In the following sections, a couple of examples of paradoxes in science and logics are presented. Each of them proves that philosophy and psychology are in the foundations of computing science.

5.1 A Paradox Example – Analogy

This section proves by contradiction that mathematical logics is not the only foundation of computing science.

An example of this is in the content of deductive proofs. Regardless of how mathematical and formal the problem be, what makes some given proof support some problem **X** and not some different problem **Y**, is the *analogy* which is made between a given problem and the representation of its relevant context, whether this representation is formal or not. Applied proofs are regarded as correct only if

1. The deduction is logically valid, i.e. in accordance with the deductive rules of the used logic.
2. The representation of the problem is correct and, in particular, complete where the variable is universally quantified.

The second item describes a basic analogy, which in its turn is essentially an informal concept, somewhat personal. On the other hand, this concept is fundamental in the context of applied logics and, hence, of the computability theory, and so on. As a metaphor (analogy), this resembles the incompleteness theorem, because the fact that analogy is not part of the mathematical methods implies that those methods are not sufficient in computing science. That is, analogy supports mathematics while the former is not supported by the latter. Therefore, mathematics is not sufficient in this rigorous sense, while rigor is a kind of ideal in mathematics. As a consequence of this, philosophy, psychology and other human studies which deal with analogy[7], are new components in the foundations of computing science, whereas philosophy and psychology support mathematics and computing science. In this way, mathematics in computing science is a fundamental tool for letting the involved concepts be precise and clear.

5.2 Another Paradox Example – Induction

The previous example applies to the content of proofs. In contrast, the present example applies to the human inference forming generalizations, which one refers to as induction[16,17]. As it is well known, both deductive logics and sciences, in a rigorous sense, reject not only analogy but also induction. By the way, both analogy and induction are referred to by philosophy as *irrational.* The fact is that, in rigorous way, both science and deductive logics reject the inductive method, in particular because its validity is not universally acceptable. Nonetheless, science itself works in the presence of panels (i.e. committees) in Master or PhD courses/research, in selections for professorship, and considerations for publications. In the end, however mathematical the particular subject is, it depends on the present induction. For the same level of reputation, even taking it as the minimum, we humans tend to *believe* that the more the number of examiners, the more accurate the result is. However, this exemplifies that logics and science work by using some method rejected by them, which constitutes a paradox at the practical level. Establishments can rely upon the intellectual attributes of the examiners, but *belief*[37], as a synthetic concept, cannot be supported by logics, mathematics nor sciences in some rigorous sense.

Moreover, between us, the above proof (as well as the previous one) not only applies to computing science... It shows how logics itself is still part of philosophy!

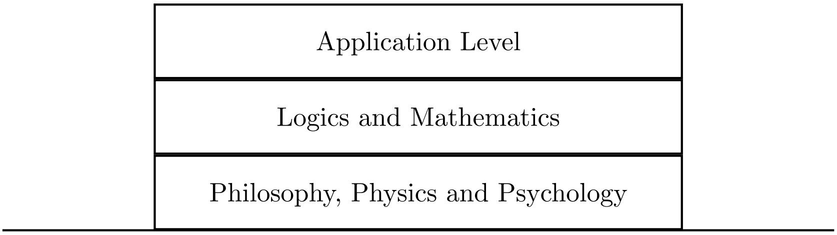
This is the most general meaning of the above deduction, whereas one of the consequences is that philosophy is in computer science, in the foundations.

The present author observes from both examples, above, that although the synthetic and human aspects are not entirely supported by logics, the latter is always supported by human beings in a synthetic way. Note that logics is traditionally philosophy. This shows the hierarchy of the large subjects inside the foundations of computing science, for philosophy and psychology can support the others. From this, different *theories* of computing science with synthetic notions can exist instead of a unique, mathematical and analytical theory. The novel area is called *philosophy of computing science.*

6 Conclusions

This paper is a synthetic discussion, on philosophy of computing science, presenting some of the links between concepts, trying to describe a semantic network. Because of this, given its purpose, it was not meant to be like almost all scientific papers in computing science, where there exists a precise focus on a particular and analytical subject.

Logics, in a rigorous sense, is not the only foundation for philosophy, psychology, computing science etc. More than this, it was shown that there can exist the following hierarchy of subjects in computing science:



Computing science has the bizarre characteristic of being both exact and philosophy for being related to the reality. For these two facts, methods rejected by science that are applied in the daily life should be considered. In the above example of committee, in order to make the applied method be consistently scientific, one should consider not only the object, the criteria and grades, but also other variables of the real world, such as the names of the examiners, which in turn ought to be public. In this case, the scientific knowledge would necessarily be referred to as something broader.

It is known that programming is one of the key subjects in computing science. However, if one observes the somewhat empirical characteristics in programming, the ideas contained here will become clearer. Yet, there seems to be nothing wrong in the way that programmers still work, and will probably continue doing. Furthermore, although one can prove that a given program is correct, there can be proofs of proofs of program correctness etc. Programs are either correct or not with respect to some representation of a relevant model from the real world. Therefore, although there are programming techniques including those suggested and imposed by programming languages, programming is a complex task that requires some very basic synthetic skills.

My classification is itself synthetic and, as such, can be neither proved nor refuted. However, exceptions exist. Other synthetic subjects can be existentially proved, and some such subjects can be equally refuted. The classification here is essentially based on intuition, analogy and induction (not only because of any possible contribution to three different areas of research, but also because of its inductive nature, I had to present sample applications to programming languages, knowledge representation, and foundations of computing science), as well as many observations on the real world. On the other hand, such a classification is not scientific by its nature, only philosophic, but philosophy and science go together. The term *science* has a sense weaker than *computer science* has had since Gödel and Turing.

Finally, the two-axis diagram reflects only some particular philosophical view. Because of this, I do not expect that it can be used as a universal tool, nor accepted by the whole community as valid. However, sections 5.1 and 5.2, in a sense, show that the diagram somehow can be useful, by taking two concepts classified in ϕ, and because, according to the corresponding classification, logics belongs to the class π. Naturally, the classification can be used by others who like it.

Acknowledgement

I want to thank the people from The Trinity College, University of Dublin, for having approved my research towards the Doctor in Philosophy.

References

1. S. Abramsky. *Handbook of Logic in Computer Science*, volume 3: Semantic Structures, chapter Domain Theory, pages 1–168. Oxford University Press, 1994.
2. L. Allison. *A Practical Introduction to Denotational Semantics.* Number 23 in Cambridge Computer Science Texts. Cambridge University Press, 1986. Reprinted 1995.
3. R. Amadio and P.-L. Curien. *Domains and Lambda-Calculi.* Number 46 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.
4. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
5. W. Bibel and E. Eder. *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1: Logical Foundations, chapter Methods and Calculi for Deduction, pages 67–182. Oxford University Press, 1993.
6. S. Blackburn. *The Oxford Dictionary of Philosophy.* Oxford University Press, 1994.
7. D. Burrell. *Analogy and Philosophical Language.* Yale University Press, 1973.
8. A. Chagrov and M. Zakharyaschev. *Modal Logic*, volume 35, chapter Complexity Problems. Oxford University Press, 1997.
9. M. Dummett. *The Philosophy of Mathematics*, chapter The Philosophical Basis of Intuitionistic Logic. Oxford Readings in Philosophy. Oxford University Press, 1996.
10. R. L. Epstein. *The Semantics Foundations of Logic*, chapter Intuitionism, page 277. Oxford University Press, 1995.
11. J. U. Ferreira. The plain www page.
    URLs http://www.ufba.br/˜plain or http://www.cs.tcd.ie/˜ferreirj, 1996–.
12. J. U. Ferreira. *uu* for programming languages. *ACM SIGPLAN Notices*, 35(8):20–30, August 2000.
13. J. U. Ferreira. *Computation in the Real World: foundations and programming languages concepts*, chapter 9 ***uu*** in Globallog. PhD thesis, The Trinity College, University of Dublin, 2001.

14. J. U. Ferreira. *Computation in the Real World: foundations and programming languages concepts*. PhD thesis, The Trinity College, University of Dublin, 2001.
15. P. Forrest. *The Dynamics of Belief: A Normative Logic*. Philosophical Theory. Basil Blackwell, 1996.
16. D. Gillies. *Artificial Intelligence and Scientific Method*, chapter 1 The Inductivist Controversy, or Bacon versus Popper, pages 1–16. Oxford University Press, 1996.
17. D. Gillies. *Artificial Intelligence and Scientific Method*, chapter 5 Can there be an inductive logic?, pages 98–112. Oxford University Press, 1996.
18. J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1993.
19. C. Gunter and D. Scott. *Handbook of Theoretical Computer Science*, volume B Formal Models and Semantics, chapter 12 Semantic Domains, pages 633–674. The MIT Press/Elsevier, 1990.
20. C. A. Gunter. *Semantics of Programming Languages: structures and techniques*. Foundations of Computing Series. The MIT Press, 1992.
21. I. Hacking. *What Is a Logical System?*, chapter What Is Logic?, pages 1–33. Number 4 in Studies in Logic and Computation. Claredon Press, Oxford University, 1994.
22. C. G. Jung and A. Storr. *Jung: Selected Writings*, chapter Psychological Typology (1936), pages 133–146. Fontana Pocket Readers. Fontana Paperbacks, second edition, 1986. Selected and Introduced by Anthony Storr.
23. I. Kant and translation by Norman K. Smith. *Immanuel Kant's Critique of Pure Reason*. Macmillan Press Ltd, 1787, 1929.
24. F. N. Kerlinger and H. B. Lee. *Foundations of Behavioral Research*. Harcourt College Publishers, fourth edition, 2000.
25. J. W. Klop. *Handbook of Logic in Computer Science*, volume 2 Background: Computational Structures, chapter Term Rewriting Systems, pages 1–116. Oxford University Press, 1992.
26. B. Kosko. *Fuzzy Thinking: The New Science of Fuzzy Logic*. HarperCollinsPublishers, Flamingo, 1994.
27. S. M. Lane. *Categories for the Working Mathematician*. Graduate texts in mathematics. Springer, second edition, 1998. Previous edition: 1971.
28. F. W. Lawvere and S. H. Schanuel. *Conceptual Mathematics: a first introduction to categories*. Cambridge University Press, 1997.
29. P. Maddy. *The Philosophy of Mathematics*, chapter Perception and Mathematical Intuition. Oxford Readings in Philosophy. Oxford University Press, 1996.
30. J. C. Mitchell. *Foundations for Programming Languages*. Foundations of Computing. The MIT Press, 1996.
31. C.-H. L. Ong. *Handbook of Logic in Computer Science*, volume 4: Semantic Modelling, chapter Correspondence Beteen Operational and Denotational Semantics: the full abstraction problem for PCF, pages 269–356. Oxford University Press, 1995.
32. C. Parsons. *The Philosophy of Mathematics*, chapter Mathematical Intuition. Oxford Readings in Philosophy. Oxford University Press, 1996.
33. K. Popper. *The Logic of Scientific Discovery*. Karl Popper, 1972.
34. B. Russell. *History of Western Philosophy*, volume III, part 2, chapter 18, The Romantic Movement, pages 651–659. Routledge, 1946. Edition published in 2000.
35. B. Russell. *History of Western Philosophy*, volume I, part 2, pages 101–226. Routledge, 1946. Edition published in 2000.
36. D. A. Schmidt. *The Structure of Typed Programming Languages*. Foundations of Computing Series. The MIT Press, 1994.

37. M. Swain, editor. *Induction, Acceptance, and Rational Belief.* D. Reidel Publishing Company, 1970.
38. R. D. Tennent. *Semantics of Programming Languages.* PHI series in computer science. Prentice Hall, Inc., 1991.
39. R. D. Tennent. *Handbook of Logic in Computer Science*, volume 3: Semantic Structures, chapter Denotational Semantics, pages 170–322. Oxford University Press, 1994.
40. J. van Heijenoort. *From Frege to Gödel.* Harvard University Press, 1967.
41. R. F. C. Walters. *Categories and Computer Science.* Cambridge Computer Science Texts. Cambridge University Press, 1991.

Toward a Structure Domain Interoperability Space

Claus Atzenbeck, Uffe K. Wiil, and David L. Hicks

Department of Software and Media Technology
Aalborg University Esbjerg
Niels Bohrs Vej 8, 6700 Esbjerg, Denmark
{atzenbeck,ukwiil,hicks}@cs.aue.auc.dk

Abstract. 'Structure domain interoperability' is a rather new field of research. This paper focuses on a possible way to describe this term as well as to show different ways of how current projects support interoperability. It will be shown that there are different approaches. This fact leads to the desire to define an interoperability space which allows to put all approaches in relation to each other.

1 Introduction

When Vannevar Bush explained his idea of "building a trail of ... items" [4], the basic notion of hypertext has been explicitly mentioned, although the term was defined almost two decades later [18]. From then, hypertext was referring to something what we call *navigational hypertext* or *navigational structure* nowadays. This modification became necessary, because many other so called hypertext domains were brought up. Examples include among others *spatial structures* [13,14], *taxonomic structures* [26], *hyperfiction* [3,12], and *argumentation support* [5,6].

The increasing amount of different domains which became topics of the hypertext community has lead to a lack of model support: Navigational hypertext models, like Dexter [10], were not able to support additional aspects of other domains than the navigational one. To solve this problem, component-based open hypermedia systems (CB-OHS) [31] and later structural computing (SC) environments [20,11], which handle structures as first-class objects, have been build. It became possible to implement several arbitrary structure servers within the same framework.

However, an open number of different structure servers within the same environment does not necessarily mean that there is much direct or indirect communication among them; they can be isolated. Researchers came up with different solutions to solve this lack of interoperability. Some of them will be discussed in Section 3, but before that, a brief description of 'structure domain' will be given.

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 66–71, 2004.

2 What Is a Structure Domain?

Certainly, there is still a lot of work to do in reaching a common understanding of 'structure' in general as well as 'structure domain' in particular [21]. One good starting point for a definition is: "A 'hypermedia domain' is defined by a coherent set of abstractions solving a particular organization problem" [25]. Considering that 'hypermedia domain' – at least with its common connotation – does not fit for all supported structure domains in a SC environment of which can be thought of, the original term should be changed to 'structure domain'. Also, the term 'abstractions' in this context points to structure abstractions rather than arbitrary abstractions. Therefore a structure domain should be considered as a coherent set of *structure* abstractions.

To finish this section with a summary of how 'structure domain' is being used in this paper: A structure domain is described as a coherent set of structure abstractions solving a particular organization problem.

3 Ways of Interoperability

The next sections will show different approaches of structure domain interoperability. The conclusion will be that they attempt to solve the same problem from different sides. This recognition will lead to a scheme in the future which allows to put all different kinds of domain interoperability in relation to each other.

3.1 Application Compatibility

This 'type' of interoperability can be observed for different domains. For example, a person uses a mindmap application – which basically represents knowledge as tree structure – for brainstorming and exports the result as linked HTML pages in order to support common web browsers (navigational structure domain). This is only possible if there is a well-defined interface between both applications.

3.2 Representation at the Same GUI

Another possibility for supporting interoperability is to create a single GUI which can handle several structure domains at the same time. Examples include all applications which offer additional structure features, like mindmap applications which allow to draw links from one branch to another (navigational links) or use color or pictures (spatial attributes). Examples also cover many so called spatial hypertext applications like *VIKI* [14], *Visual Knowledge Builder (VKB)* [24], *Manufaktur* [17], or *Topos* [9], which support mainly spatial structures, but also offer container objects for representing hierarchies and navigational linkage.

Some other approaches yield toward a complete merge of different structures, for instance Eastgate's *Tinderbox* [8]. This application does not have an obvious prior domain, but focuses on spatial, hierarchical, and navigational structures as well as on metadata. All of them can be made visible with one single application.

Another project, done by Fraunhofer IPSI Institute, aims to combine navigational, spatial, taxonomic, and workflow structures within one single user interface [27].

3.3 Wrapper Services

Wrappers support domain interoperability, because they allow non-native services to connect to any supported service, even if they belong to other structure domains. In principle, wrappers are capable of wrapping any module, including services at the middleware and backend levels. For instance, *Construct*'s wrappers make it possible for *Emacs* or *Netscape Navigator* (frontend applications) to connect to navigational or metadata services (structure components) [31,30,19]. Another example is the "HyperDisco–Chimera interoperability experiment" [32], which uses a wrapper to connect *HyperDisco* (linking/storage protocol) and *Chimera* (linking protocol).

3.4 Structure Transformation

Interoperability can also be supported by transforming structures. This is what *Structure Definition Language (SDL)* and *Structure Transformation Language (STL)* is for [1]. This approach recently has found its incarnation as a SC environment called *Themis* [2]. It offers templates for instantiating structures.

3.5 'Super Structure'

An approach which focuses on creating a single model, supporting taxonomic, spatial, and navigational structures, is the *Fundamental Open Hypermedia Model (FOHM)* [16,15]. *FOHM*'s main advantage is to have a high level of interoperability among those three structure domains – even associations between them are possible –, but it recently has been criticized, because new structure domains cannot be added easily as it is possible with *Construct*, *Callimachus*, or *Themis* [25,2]. This restricts interoperability to services 'outside' *FOHM*.

3.6 Structure Linking

Modularized services have advantages regarding exploration of interactions and independence [31]. With respect to this, it can be argued that structure servers within one single SC environment should not be able to do anything which belongs to another structure server within the same framework: Tasks of structure services should be uniquely assigned. However, a user might want to connect structures which are handled by several separate services. This can only be supported if those structure services can communicate among each other. Because structures are first-class objects, any part of a SC system should be able to manipulate them. Therefore using one structure to associate any other structure should not be a problem.

When would this be useful? For example, if a spatial structure needs to be arranged within a hierarchy or if metadata has to be assigned to a navigational link using the system's metadata structure service.

3.7 Standards Addressing Interoperability

When discussing interoperability it usually comes to the question of interoperability addressing standards. They make it easy to connect different services, including structure servers. One standardization instance is the Open Hypermedia Working Group (OHSWG) [7], which has defined a standard for linking frontend applications and structure servers: The *Open Hypermedia Protocol (OHP)* [23]. "It is a nature next step in the standardization process that allows the OHSWG to broaden its standardization and interoperability effort to cover all layers of open hypermedia systems and not just the interfaces between applications and structure services" [31].

4 Future Work

This paper has shown that there are many projects working on different interoperability approaches, but there is no general scheme which places them within a framework. *Interoperability space* would be a possible name for it. Only limited attempts have been made for an interoperability space [31], but there are several similar approaches for building frameworks for other fields of hypermedia [29,28,32,22]. They may give some further inputs.

After classifying known types of interoperability and placing them in relation to each other, the new created space can be used to point out differences among them as well as advantages and disadvantages. Furthermore this framework will offer a scheme to SC developers which lets them find the appropriate method for their task. There is also the chance that new kinds of domain interoperability will be found.

The interoperability space will be proved by data gained by a user survey which observes how people interact with structures from various domains. Also current interoperability efforts will be placed within the space to check whether it can be used without restrictions. The correctness of the created space can only be claimed if all tests turn out positive and only until there is no new described type which does not fit into the scheme.

References

1. K. M. Anderson. Structural computing requirements for the transformation of structures and behaviors. In S. Reich and K. M. Anderson, editors, *Open Hypermedia Systems and Structural Computing. 6th International Workshop, OHS-6. 2nd International Workshop, SC-2. San Antonio, TX, June 3, 2000*, volume 1903 of *Lecture notes in computer science*, pages 140–146. Springer, 2000.

2. K. M. Anderson, S. A. Sherba, and W. V. Lepthien. Structural templates and transformations: the themis structural computing environment. *Journal of Network and Computer Applications*, 26(1):47–71, 2003.
3. J. D. Bolter and M. Joyce. Hypertext and creative writing. In *Proceeding of the ACM conference on Hypertext*, pages 41–50. ACM Press, 1987.
4. V. Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 7 1945.
5. J. Conklin and M. L. Begeman. gibis: a hypertext tool for team design deliberation. In *Proceeding of the ACM conference on Hypertext*, pages 247–251. ACM Press, 1987.
6. J. Conklin and M. L. Begeman. gibis: a hypertext tool for exploratory policy discussion. In *Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, pages 140–152. ACM Press, 1988.
7. H. C. Davis, D. E. Millard, S. Reich, N. Bouvin, K. Grønbæk, P. J. Nürnberg, L. Sloth, U. K. Wiil, and K. Anderson. Interoperability between hypermedia systems: the standardisation work of the ohswg. In *Proceedings of the tenth ACM Conference on Hypertext and hypermedia: returning to our diverse roots*, pages 201–202. ACM Press, 1999.
8. Eastgate Systems. *Tinderbox^{TM} for Macintosh v. 2.0. User's Manual & Reference*, 2003.
9. K. Grønbæk, P. P. Vestergaard, and P. Ørbæk. Towards geo-spatial hypermedia: Concepts and prototype implementation. In *Proceedings of the thirteenth conference on Hypertext and hypermedia*, pages 117–126. ACM Press, 2002.
10. F. Halasz and M. Schwartz. The dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, 2 1994.
11. D. L. Hicks and U. K. Wiil. Searching for revolution in structural computing. *Journal of Network and Computer Applications*, 26(1):27–45, 2003.
12. M. Joyce, N. Kaplan, J. McDaid, and S. Moulthrop. Hypertext, narrative, and consciousness. In *Proceedings of the second annual ACM conference on Hypertext*, pages 383–384. ACM Press, 1989.
13. C. C. Marshall, F. G. Halasz, R. A. Rogers, and W. C. Janssen. Aquanet: a hypertext tool to hold your knowledge in place. In *Proceedings of the third annual ACM conference on Hypertext*, pages 261–275. ACM, ACM Press, 1991.
14. C. C. Marshall, F. M. Shipman, and J. H. Coombs. Viki: spatial hypertext supporting emergent structure. In *Proceedings of the 1994 ACM European conference on Hypermedia technology*, pages 13–23. ACM Press, 1994.
15. D. E. Millard. Discussions at the data border: from generalised hypertext to structural computing. *Journal of Network and Computer Applications*, 26(1):95–114, 2003.
16. D. E. Millard, L. Moreau, H. C. Davis, and S. Reich. Fohm: A fundamental open hypertext model for investigating interoperability between hypertext domains. In *Proceedings of the eleventh ACM on Hypertext and hypermedia*, pages 93–102. ACM, ACM Press, 2000.
17. P. Mogensen and K. Grønbæk. Hypermedia in the virtual project room – toward open 3d spatial hypermedia. In *Proceedings of the eleventh ACM on Hypertext and hypermedia*, pages 113–122. ACM Press, 2000.
18. T. H. Nelson. Complex information processing: a file structure for the complex, the changing and the indeterminate. In *Proceedings of the 1965 20th national conference*, pages 84–100. ACM Press, 1965.
19. Y. Neveu, Y. Guervilly, U. K. Wiil, and D. L. Hicks. Providing metadata on the world wide web. Technical Report CSE-01-01, Aalborg University Esbjerg, 2001.

20. P. J. Nürnberg, J. J. Leggett, and E. R. Schneider. As we should have thought. In *Proceedings of the eighth ACM conference on Hypertext*, pages 96–101. ACM, ACM Press, 1997.
21. P. J. Nürnberg and M. M. C. Schraefel. Structural computing and its relationships to other fields. In S. Reich, M. M. Tzagarakis, and P. M. E. De Bra, editors, *Hypermedia: Openness, Structural Awareness, and Adaptivity. International workshops OHS-7, SC-3, and AH-3 (Århus, Denmark, August 14–18, 2001)*, pages 183–193. Springer, 2002.
22. K. Østerbye and U. K. Wiil. The flag taxonomy of open hypermedia systems. In *Proceedings of the the seventh ACM conference on Hypertext*, pages 129–139. ACM Press, 1996.
23. S. Reich, U. K. Wiil, P. J. Nürnberg, H. C. Davis, K. Grønbæk, K. M. Anderson, D. E. Millard, and J. M. Haake. Addressing interoperability in open hypermedia: The design of the open hypermedia protocol. *The New Review of Hypermedia and Multimedia*, 5:207–248, 1999.
24. F. M. Shipman, H. Hsieh, P. Maloor, and J. M. Moore. The visual knowledge builder: a second generation spatial hypertext. In *Proceedings of the twelfth ACM conference on Hypertext and Hypermedia*, pages 113–122. ACM, ACM Press, 2001.
25. M. Tzagarakis, D. Avramidis, M. Kyriakopoulou, M. M. C. Schraefel, M. Vaitis, and D. Christodoulakis. Structuring primitives in the callimachus component-based open hypermedia system. *Journal of Network and Computer Applications*, 26(1):139–162, 2003.
26. H. Van Dyke Parunak. Don't link me in: set based hypermedia for taxonomic reasoning. In *Proceedings of the third annual ACM conference on Hypertext*, pages 233–242. ACM Press, 1991.
27. W. Wang and A. Fernández. A graphical user interface integrating features from different hypertext domains. In S. Reich, M. M. Tzagarakis, and P. M. E. De Bra, editors, *Hypermedia: Openness, Structural Awareness, and Adaptivity. International workshops OHS-7, SC-3, and AH-3 (Århus, Denmark, August 14–18, 2001)*, pages 141–150. Springer, 2002.
28. E. J. Whitehead. Design spaces for link and structure versioning. In *Proceedings of the twelfth ACM conference on Hypertext and Hypermedia*, pages 195–204. ACM, ACM Press, 2001.
29. U. K. Wiil. A framework for classifying extensiblility mechanisms in hypermedia systems. *Journal of Network and Computer Applications*, 24(1):7–18, 2001.
30. U. K. Wiil and D. L. Hicks. Providing structural computing services on the world wide web. In S. Reich, M. M. Tzagarakis, and P. M. E. De Bra, editors, *Hypermedia: Openness, Structural Awareness, and Adaptivity. International workshops OHS-7, SC-3, and AH-3 (Århus, Denmark, August 14–18, 2001)*, pages 160–182. Springer, 2002.
31. U. K. Wiil, D. L. Hicks, and P. J. Nürnberg. Multiple open services: A new approach to service provision in open hypermedia systems. In *Proceedings of the twelfth ACM conference on Hypertext and Hypermedia*, pages 83–92. ACM, ACM Press, 2001.
32. U. K. Wiil and K. Østerbye. Using the flag taxonomy to study hypermedia system interoperabilty. In *Proceedings of the ninth ACM conference on Hypertext and hypermedia : links, objects, time and space – structure in hypermedia systems*, pages 188–197. ACM Press, 1998.

An Efficient E-mail Monitoring System for Detecting Proprietary Information Outflow Using Broad Concept Learning

Byungyeon Hwang[1] and Bogju Lee[2]

[1] Department of Computer Engineering, Catholic University, Korea
byhwang@catholic.ac.kr
[2] Department of Computer Engineering, Dankook University, Korea
blee@dankook.ac.kr

Abstract. The automatic text categorization (ATC) is an emerging field that is technically based on the machine learning and information retrieval. Traditional machine learning usually builds a model that captures the behavior of the 'concept' from the training instances. Following this context, the ATC learns the concept from text data such as digital text documents, web pages, or e-mail texts. In this paper, we categorize the concepts that have been dealt by ATC into two groups: broad concepts and narrow concepts. Broad concepts are difficult to learn than narrow concepts. We propose a multi model approach where multiple local models are constructed to learn the broad concepts. Some experimental results showed that the multi model approach turned out to be effective in learning the confidentiality of company.

1 Introduction: Learning the Broad Concepts

The automatic text categorization (ATC) is an emerging field that is technically based on the machine learning and information retrieval. Traditional machine learning algorithms as well as new text mining algorithms such as the latent semantic indexing [1], the naïve Bayes [2], the neural networks [3], the k-nearest neighbor [2], the decision tree [4], the inductive rule learning [5], the Rocchio [1], [6], and the support vector machines (SVMs) [2] have been used. The techniques to manipulate the text are borrowed from the information retrieval area, which include the term weighting, the bag of words approach, and the vector space model, and more.

Traditional machine learning usually builds a model that captures the behavior of the target "concept" from the training instances. The model could be the form of rules, networks, or trees. Following this context, the ATC learn the subjects (i.e., concepts) from text data that exist in the form of digital text documents, web pages, or e-mails.

The ATC has been successfully applied to various text categorization works. Some researches applied text categorization techniques to e-mail message categorization problem [7], [8]. This used machine learning algorithms for personal information

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 72–78, 2004.

agent, which categorizes incoming e-mails by predefined set of subjects for a person. Also another application of ATC with e-mails was proposed in [9] that learn authorship identification and categorization. SVM was used in this research and showed promising results. Another problem domain, customer e-mail routing, can be found in [10], that used decision tree to classify customer e-mails by predefined subjects such as address change and check order.

There are also many attempts to use text categorization to eliminate junk e-mails [11], [12]. In these researches text categorization techniques are used to filtering out unwanted e-mails from user's point of view. This is close to the Newsweeder system [13] that aims to learn the interestingness of a user. In [14], an e-mail monitoring system is presented. The ATC is successfully applied to detect the outflow of "confidential" documents of a company. In this task the SVM learns the confidentiality of company, given training documents.

Looking into the various subjects or targets to be learned, we can categorize the subjects into two groups: one that is narrow (e.g., news articles about company merge and acquisition; news about stocks; customers' change of address; and so on) and the other that is broad (e.g., junk e-mails; news which interest a specific user; confidentiality of a company). Observing the training data of these two groups, there is a clear difference. The terms/words representing the broad concepts never appear in the training data (e.g., junk e-mails never say they are junk e-mails; interesting news never say it is interesting to a specific user; a confidential document of a company never mention it is confidential). On the other hand, narrow concepts usually appear in the training data explicitly: e.g., 'merge' and 'acquisition' often appear in the news articles about the company M&A; most of the stock news contain the 'stock' word. It is easily observed that broad concepts are difficult to learn than narrow concepts. The target broad concept usually consists of various "local" concepts: e.g., junk e-mails are very diverse in terms of the subjects. Confidential documents in an organization are diverse too. They are from many divisions and departments. Since learning the broad concepts has different characteristics, the ATC approach for the broad concepts should be different. We would like to propose a general approach to learn the broad concepts.

Section 2 describes the existing approaches to learn the broad concepts. Our approach to learn the broad concepts is presented in Section 3. Section 4 demonstrates some experimental results. Section 5 discusses the conclusion.

2 Existing Approaches to Learn the Broad Concepts

The approaches used in learning the broad concept tasks and listed in the previous section are analyzed here. These include the junk e-mail filtering [15], Newsweeder system [13], and learning the company confidentiality [14]. The approach used in the junk e-mail filtering is nothing special compared with general ATC tasks. When collecting training data, an AT&T staff member distinguished spam mails and non-spam mails from his personal judge. The SVM and the boosting achieved good results with error rate of about 0.01 to 0.03.

The Newsweeder let the user rate the usenet article as he/she reads. After training using the naïve Bayes, the system presents the top 10% of automatically rated article. It achieved about 60% of accuracy.

The approach used in learning the confidentiality of a company [14] is also similar to the junk mail filtering. The confidential documents go to one pool (i.e., positive training data) although they are from several divisions and departments. Negative training data consist of non-confidential documents appointed by the company and the general web documents. In spite of using this simple approach, it showed relatively good performance. Fig. 1 depicts the e-mail monitoring system that employs this learning task as a component.

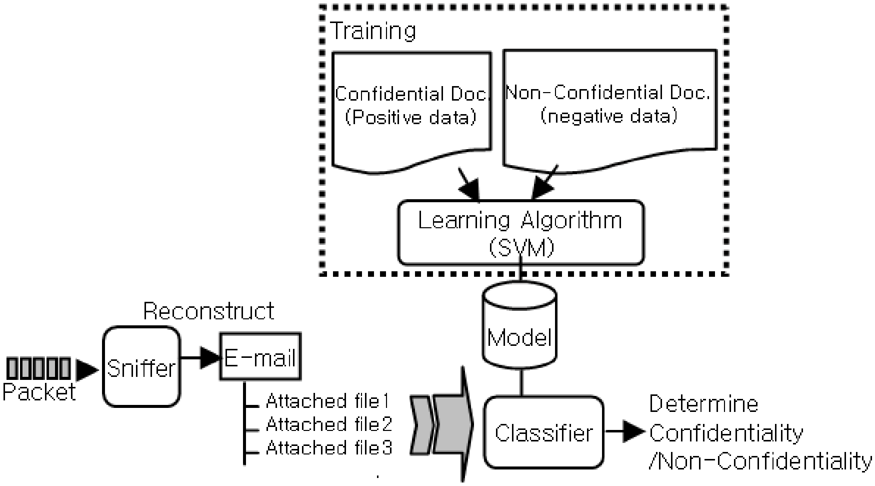


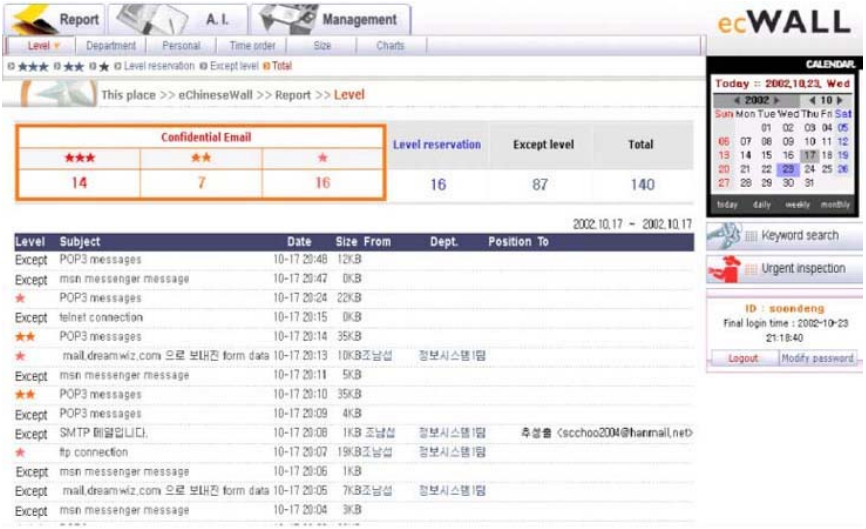
Fig. 1. The e-mail monitoring system that detects the outflow of confidential documents in a company. The system constructs a single model representing the concept of confidentiality of a company.

Fig. 2 is a sample report screen of the monitoring result. Through a user-friendly interface, the managers can register the confidential documents as the training data, view and drop the registered documents, execute the learning phase, and query the monitoring results. The report in default form shows the monitored e-mails classified by confidentiality level, creation time, size, and so on.

The common approach that can be found in the three tasks above is that it never tries to separate the local concepts from the target concept, although there does exist. It simply builds a "single model" for the target broad concept. In the next section we present the "multi model" approach for the broad concept.

3 Multi-model: Our Approach to Learn the Broad Concepts

Broad concepts consist of various local concepts. For example, junk e-mails to a person may be grouped by adult e-mails, advertisements, and other entertainment e-



Confidential Email			Level reservation	Except level	Total
★★★	★★	★			
14	7	16	16	87	140

2002.10.17 – 2002.10.17

Level	Subject	Date	Size	From	Dept.	Position	To
Except	POP3 messages	10-17 20:48	12KB				
Except	msn messenger message	10-17 20:47	0KB				
★	POP3 messages	10-17 20:24	22KB				
Except	telnet connection	10-17 20:15	0KB				
★★	POP3 messages	10-17 20:14	35KB				
★	mail.dreamwiz.com 으로 보내진 form data	10-17 20:13	10KB	조남섭	정보시스템1팀		
Except	msn messenger message	10-17 20:11	5KB				
★★	POP3 messages	10-17 20:10	35KB				
Except	POP3 messages	10-17 20:09	4KB				
Except	SMTP 메일입니다.	10-17 20:08	1KB	조남섭	정보시스템1팀		추성출 <scchoo2004@hanmail.net>
★	ftp connection	10-17 20:07	19KB	조남섭	정보시스템1팀		
Except	msn messenger message	10-17 20:06	1KB				
Except	mail.dreamwiz.com 으로 보내진 form data	10-17 20:05	7KB	조남섭	정보시스템1팀		
Except	msn messenger message	10-17 20:04	3KB				

ID : soondeng
Final login time : 2002-10-23 21:18:40

Fig. 2. A sample report screen of monitoring result.

mails that he/she does not like. A person's interestingness obviously consists of many subjects. The confidentiality of a company also consists of multiple subjects.

One of the nice things in learning the confidential documents is that the training data come in with already well-grouped fashion. There are multiple divisions and departments in a company and the confidential documents for a division/department can be regarded to represent one local concept. Our idea is to build a model for each local concept instead of a single model for the whole concept. It is depicted in Fig. 3. The multi model in the figure tries to build three local models where each model uses confidential documents from a department. Note that negative data for each model include the confidential documents from the other departments.

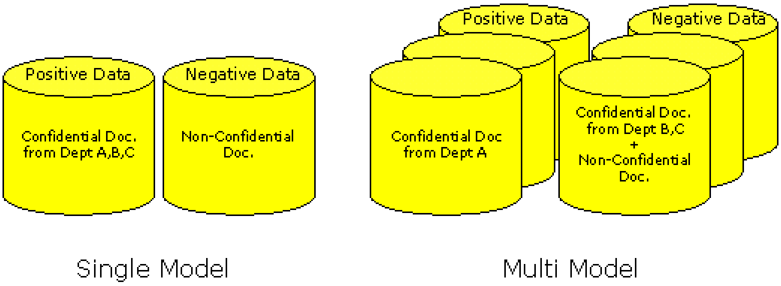

Positive Data
Confidential Doc. from Dept A,B,C
Negative Data
Non-Confidential Doc.
Single Model
Positive Data
Confidential Doc from Dept A
Negative Data
Confidential Doc. from Dept B,C + Non-Confidential Doc.
Multi Model

Fig. 3. Single model and multi model.

Once the multi models are constructed, the system goes to the classification phase. When deciding the confidentiality of a document in this phase, a testing document is predicted against all the constructed local models. Even single detection from a local model is regarded as corresponding to the target concept (i.e., confidential).

4 Experimental Results

In order to make the experiment as real as possible, we chose real companies: a telecommunication company, a finance company, and a medicine company. The performance comparisons of the single model and multi model for each different company are presented. Table 1 below shows the experimental result of the multi model and single model using the data from a telecommunication company. The company has five different divisions that have 18, 33, 14, 11, and 22 confidential documents, respectively. Note again that the negative data consists of the confidential documents from other divisions and the general web documents. For example, 230 negative data for the division A include 80 (=33+14+11+22) confidential documents from the divisions B, C, D, and E. The rest 150 (=230-80) documents are general web documents. The leave-one-out performance measure is used here. The performance average for the multi model indicates that multi model performs a little bit better than the single model.

Table 1. Performance result of the multi model and single model with a telecommunication company.

Multi model	Training set			Leave-one-out		
Division	Positive data	Negative data	Total	Error rate	Precision	Recall
A	18	230	248	11.1%	100%	88.9%
B	33	215	248	0%	100%	100%
C	14	234	248	14.3%	100%	85.7%
D	11	237	248	16.7%	100%	83.3%
E	22	226	248	0%	100%	100%
Avg.	27	228.4	248	8.4%	100%	91.6%
Single model	98	150	248	8.9%	100%	91.1%

Table 2 below shows the experimental result for the medicine company that has three different divisions. This time the multi model performs better than the single model.

Table 2. Performance result of the multi model and single model with a medicine company.

Multi model	Training set			Leave-one-out		
Division	Positive data	Negative data	Total	Error rate	Precision	Recall
A	36	187	223	0%	100%	100%
B	21	202	223	9.1%	100%	90.9%
C	16	207	223	12.5%	100%	87.5%
Avg.	24.3	198.7	223	7.2%	100%	92.8%
Single model	73	150	223	11.1%	100%	88.9%

Table 3 below shows the experimental result for the finance company that has also three different divisions. This time again the multi model performs a little bit better than the single model.

Table 3. Performance result of the multi model and single model with a finance company.

Multi model	Training set			Leave-one-out		
Division	Positive data	Negative data	Total	Error rate	Precision	Recall
A	51	185	236	0%	100%	100%
B	13	223	236	16.7%	100%	83.3%
C	22	214	236	9.1%	100%	90.9%
Avg.	28.7	207.3	236	8.6%	100%	91.4%
Single model	86	150	236	9.5%	100%	90.5%

5 Conclusions

In this paper the automatic text classification tasks are classified into two groups: the tasks of learning broad concepts and those of learning narrow concepts. The characteristics of learning broad concepts are studied also. Since the broad concepts have different characteristics and are usually difficult to learn, the different approach called multi model approach is proposed. In this approach, multiple local models are constructed to learn the broad concepts. Task of learning the confidentiality of a company is chosen for the experiment. One of the nice things in the problem is that the local concepts are already given according to the divisions or departments. The experimental results showed that the multi model approach turned out to be effective in learning the confidentiality of company.

For the problems that cannot partition the local concepts easily, e.g., junk mail filtering or user's interestingness, further research for the multi model is needed. One possibility is to cluster the training data first to obtain the local concepts and then construct the local models for the local concepts.

Acknowledgements

This study was supported by the Research Fund 2003 of the Catholic University of Korea.

References

1. Hull, D.: Improving Text Retrieval for the Routing Problem Using Latent Semantic Indexing. Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (1994)

2. Joachims, T.: Text Categorization with Support Vector Machines - Learning with Many Relevant Features. Proc. of the European Conference on Machine Learning, Springer (1998)
3. Weiner, E., Pedersen, J., and Weigend, A.: A Neural Network Approach to Topic Spotting. Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval (1995)
4. Cohen, W. and Singer, Y.: Context-Sensitive Learning Methods for Text Categorization. Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (1996)
5. Apte, C., Damerau, F., and Weiss, S.: Towards Language Independent Automated Learning of Text Categorization Models. Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (1994)
6. Ittner, D., Lewis, D., and Ahn, D.: Text Categorization of Low Quality Images. Proceeding of the 4th Annual Symposium on Document Analysis and Information Retrieval (1995)
7. Kim, S., Hall, W.: A Hybrid User Model in Text Categorization
8. Rennie, J.D.M.: ifile: An Application of Machine Learning to E-mail Filtering. Proc. of KDD-2000 Workshop on Text Mining (2000)
9. de Vel, O.: Mining E-mail Authorship. Proc. of KDD-2000 Workshop on Text Mining (2000)
10. Weiss, S.M., Apte, C., Damerau, F.J., Johnson, D.E., Oles, F.J., Goetz, T., Hampp, T.: Maximizing Text-Mining Performance, IEEE Intelligent Systems (1999)
11. Sahami, M., Dumais, S., Heckerman. D., Horvitz, E.: A Bayesian Approach to Filtering Junk E-Mail. Proc. of AAAI-98 Workshop on Learning for Text Categorization (1998)
12. Katirai, H.: Filtering Junk E-Mail: A Performance Comparison between Genetic Programming and Naïve Bayes, Carnegie Mellon University (1999)
13. Lang, K.: Newsweeder: learning to filter netnews. Proceedings of the 12th International Conference on Machine Learning. Morgan Kaufmann Publishers (1995)
14. Lee, B., Park, Y.: An E-mail Monitoring System for Detecting Outflow of Confidential Documents, Lecture Note in Computer Science, Vol. 2665. Springer-Verlag, Berlin Heidelberg New York (2003)
15. Drucker, H., Wu, D., Vapnik, V.: Support Vector Machines for Spam Categorization, IEEE Transactions on Neural Networks, Vol. 10. (1999)

Interface Design – Use of Audio as an Output

Kirstin Lyon and Peter J. Nürnberg

Department of Computer Science, Aalborg University Esbjerg
Niels Bohrs Vej 8, DK-6700 Esbjerg, Denmark
{kirstin,pnuern}@cs.aue.auc.dk

Abstract. This paper analyses a number of audio interface models that are currently in use or being developed. It describes a space for describing various models of interfaces that could be used by both visually impaired (VI) and mobile computer users. This paper is concerned only with the use of audio as an output cue.
Visualisation is an increasingly important method for people to understand complex information and to navigate around structured information. Computer-based visualisation techniques often depend almost entirely on high-resolution graphics. There are several situations in which this is insufficient.

1 Introduction

At present, sound is sometimes used in a simplistic manner within Human Computer Interaction (HCI), and has at times been overtaken by the design of complex graphics. Sound is used mainly to direct the users attention to an event. This could be, for example, an error or a program starting or finishing. Visual cues, however, generally guide a user around an interface. In the areas of mobile computing, the disappearing computer initiative and for VI users, it may be useful to increase the number and type of audio output cues being used, allowing them to navigate around a computer more efficiently [9,13]. As an increase in audio cues could help with various situations.

Douglas Engelbart's idea that organisation's work can be classified according to a three level classification scheme, levels "A", "B" and "C". "Level A" work relates directly to the work of the organisation. "Level B" work helps workers to completed level A work more efficiently. "Level C" work helps level B workers to help level A works do their job. When considered within this context, much of the research in audio interface design can be thought to be level B work.

The remainder of the paper is organised as follows: Sec. 2 describes some design considerations as well as the users who may benefit. Section 3 describes the types of audio that have been developed. Section 4 describes some of the research being undertaken. Section 5 outlines the possible axes for an auditory interface. Section 6 describes four different auditory models. Section 7 includes details of future work and Sec. 8 provides conclusions.

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 79–88, 2004.

2 Background

2.1 Design Considerations

An interface should have a simple, intuitive layout in order for a user to navigate effectively. The aims of user-interface design include [10]:

1. Time to learn Time taken for a user to become familiar with the interface.
2. Speed of performance How quickly a user can navigate once they are familiar.
3. Rate of errors by user The amount of errors occurring over a given period.
4. Retention over time How well the user remembers the interface.
5. Subjective satisfaction Is the interface liked by users?

At any given time a user should be able to find out what to do, and also know what is happening. Using real world models as interfaces may satisfy the above aims for most computer users.

In order to create an efficient audio interface, it may be necessary to create a new metaphor (a new visualisation) that does not simply mirror what happens visually and translates it into audio cues. An example could be to think of a room metaphor. That room can contain various objects such as programs, files, etc. It may also have doors to other rooms. Alternatively, another visualisation could be to think of the information being arranged similar to that of a star chart or spiders web. A 3 dimensional space where users can see "layers" of information.

2.2 Potential Users

The main benefactors of such a system would include visually impaired (VI) and mobile computer users. This would also be relevant in the disappearing and wearable computer initiatives.

Visually Impaired Users. VI users may not be able to benefit from the advanced graphics that are available today. Even though many VI users have some useful sight (around 96% '[5]), they have different needs. Carefully designed graphics may be changed in order to create a more usable interface. For example, an increase of font size or a change in colour to give maximum contrast. Care must also be taken not to overload a user, and to create the right balance between useful information and background "noise." Many guidelines exist aiding in the creation of good design.([20,11])

Mobile Computer Users. Mobile computers, such as mobile phones or PDA's, provide a small screen. Often the user will either not wish to look at the screen (e.g. when walking) or not be able to do so (e.g., no lighting available,driving etc.) In the case of the disappearing [4] or wearable computer initiative [19] there may be no screen provided at all. Information may still need to be found quickly and efficiently. The dramatically reduced or lack of screen size introduces the issue of visual clutter, and of over-loading a user. Audio could be used to increase the screen size virtually [16,17].

3 Designing Audio

Heller et al. [7] describe a framework for using various media including audio. The information from the media taxonomy that is relevant to sound is summarised below.

- *Audience* Is the metaphor appropriate to the user?
- *Discipline* Does the sound conveys too much/little information?
  Do the Sound effects convey information?
- *Interactivity* What control does the user have? (None, sequential, presentation)
  Is the user able to make predefined changes(e.g. adjust volume, change stations, fast-forward tracks, loop)?
  Can the user create sounds?
- *Quality* What is the sound quality like? (unclear-clear)
  What is the sound speed like?(too fast/slow-just right)
  How is the volume?(too loud/soft-just right)
- *Usefullness* How helpful is the audio information? (not helpful-helpful)
  What is your overall impression of the sound? (confusing-clear)
  What is your overall impression of the organisation?(disorganised-organised)
- *Aesthetics* How is the sound? (annoying - helpful)
  How is the background sound? (not pleasing - pleasing)
  How is the sound effects? (not pleasing - pleasing)
  How is the sound effects?(clear - unclear)
  Describe the sound (noisy - tuneful)
  How is the volume (too soft/loud-just right)
  what did you think of sound?

3.1 Types of Audio

Research in the area of audio interfaces has developed the concepts of *auditory icons*, *sonification* and *earcons*, as well as improvements in speech technology.

Auditory Icons. Auditory icons try to mimic real world sounds [1]. This can reduce the learning period, but may have restrictions on how each object should be mapped to a sound. Some objects and actions provide a natural mapping, whereas others are less obvious. For example, what would a “redo” action sound like?

Earcons. Earcons give each object and action a short musical sequence [3]. For example, a folder will have a particular earcon, as will the action of opening it. When the two are combined it means that a file is being opened. These tend to have a longer initial training period, as they are not related to real-world sounds, but have the advantage that the number of possible earcons are limitless.

Sonfication. Sonification gives the user information about his actions immediately [1]. The audio output may change depending on the action of the user. For example, when downloading a file, sonfication can be used to show when the action is completed. It immerses the user more in the environment, and can show the passing of time. However, it can lead to artificial sounds; and, there is no real connection between the sound to that of the data being represented.

Speech Technology. The improvements in speech technology have meant that VI users can have text read to them without having to rely on a sighted helper. This has the advantage of being a natural form of receiving information. However, it can sound unnatural, and may have difficulties when words appear that is not in its vocabulary. It also takes more time to receive the information.

4 Related Work

4.1 The vOICe – Vision Technology for the Totally Blind

This application is designed to be used with a mobile computer and tries to provide "images" to VI users [18]. It works by taking in images from a camera and translating them into soundscapes. It then sends that to the user with headphones.

Everything has a unique sound, but there are principles to how the sound is chosen. At present, brighter areas sound louder, height is indicated by pitch and a built-in colour identifier speaks out the names of the colour when it is activated. When a user understands that, it is hoped that they may be able to interact more with their environment.

4.2 Drishti: An Integrated Navigation System for Visually Impaired and Disabled

Using various technologies, Drishti [6] aims to guide a VI user around a specified area, taking into account that the quickest route may not be the safest. It uses voice recognition as input, and outputs instructions via speech technology. The user must wear a headset in order to hear the information.

4.3 ESPACE

ESPACE [14] is an entirely audio interface, and uses a trackball and number pad as input. The interface uses a bubble metaphor. Each bubble has its own unique sound and contains various objects, including sub-bubbles. When the user enters a child bubble, the parent bubble sound is still heard. This allows the user to remember where they are in relation to other files.

4.4 MP3 Player Design

This is an example of an application that uses haptic input and audio output [2]. Music players are very common mobile devices and are usually used for long periods of time while the user is doing something else. The metaphor was designed to reflect real world situations, and should make it easier to learn. Earcons were used to give feedback to the user after the actions were performed.

5 Auditory Model Space

There are many axes along which models of audio user interfaces may vary. In this section, we describe several of these. Although this list is not complete, we feel it provides a reasonable starting point for further discussions. For each axis, we briefly describe the range of possible values, and give an example of the meaning of each axis value. The axes are not presented in any particular order. For quick reference, Tab. 1 summarises the axes and their values.

Table 1. Summary of model space axes and possible values

axis name	**possible values**
Cues to user	Audio only Mixed visual/audio Mixed audio/haptic
Type of sound	Auditory icons Earcons Sonfication converting text to speech continuous Mixture
Number of Simultaneous events	Single Several
Areas of use	Information retrieval User interaction Both
Delivery of sound	Mono Stereo 3D sound

Cues to User. Content may be viewed in 2 ways, either with only audio cues, or a mixture of audio and visual cues.

– *Audio.* User interacts with their environment using audio cues only.
– *Mixed visual/audio.* User interacts with their environment using both visual and audio cues.
– *Mixed audio/haptic.* User interacts with their environment using both audio and haptic cues.

Type of Sound. Audio cues can be provided using *auditory icons*, *earcons*, *sonification text to speech*, *continuous* or a mixture.

- *Auditory icons.* Everyday sounds mapped to a computer event by an analogy with everyday sound producing events (e.g. sound of paper being thrown away when recycle bin emptied in Microsoft Windows).
- *Earcons.* A language built from short sequences of tones and associated with actions and objects.
- *Sonification.* The transformation of data by a sound generator to classical dimensions of sound such as frequency, amplitude and duration for monitoring and comprehension.
- *Converting text to speech.* Each item has text associated with it. When the users scrolls over the item or clicks on it, the text is converted into speech.
- *Continuous.* Sound is more or less constantly used.
- *Mixture.* A mixture of the above concepts may be used.

Number of Simultaneous Events. The cues given to the user may be one after the other, or at the same time.

- *One.* A maximum of one cue is given to a user at any given time.
- *Several.* A variety of cues are given to a user at any given time.

Areas of Use. An audio interface may be used for interface interaction purposes, for information retrieval purposes, or both.

- *Interface interaction.* Sound cues would reflect what is happening visually and direct a user around an interface.
- *Information retrieval.* Sound cues would be used to search for specific information.
- *Both.* Sound cues would be used to act as a user interface, and be able to search for specific information.

Delivery of Sound. This refers to the kind of sound that the user will hear.

- *Mono.* No directional sound.
- *Stereo.* User may pan from left to right.
- *3D sound.* Sound may be positioned 360 degrees around the user's head, as well as above and below. This may be achieved with using either a surround sound system with speakers, or using Head Related Transfer Function (HRTF) with headphones.

6 Populating the Model Space

In this section, we consider four models of audio interface in the light of the axes discussed above. These models have been chosen to cover most eventualities, and most of the model space described in the previous section. We also consider the success of these models in terms of increasing users' ability to interact with an interface.

6.1 Traditional Model

By "traditional model," we mean the publishing model that is exemplified by current desktop systems, e.g. Microsoft Windows. In our model space, the traditional model is defined as follows.

– *Cues to user.* This model uses mainly visual to provide cues, but has some sound available.
– *Type of sound.* Auditory icons for the most part (e.g. emptying recycle bin). Other sound does not fall into a category.
– *Number of simultaneous events.* One event at a time.
– *Areas of use.* Sound is used mainly as a warning, and to signal the beginning/end of some programs.
– *Delivery of sound.* This model uses mono.

This model uses very little sound, and the sound produced may irritate some users. In some cases computers are not available with speakers in an office environment, or the sound is muted. No real interaction is available with this level of audio. It also does not support visually impaired users who must use another application in order to interact with the operating system interface. There are difficulties with mobile computers where the space for a graphical display is considerably reduced.

6.2 ESPACE 2

This model describes the ESPACE 2 prototype system [14] as designed by Nitin Sahwney and Arthur Murphy.

– *Cues to user.* This model uses audio cues to guide the user.
– *Type of sound.* Auditory icons and continuous audio.
– *Number of simultaneous events.* Multiple events at a time.
– *Areas of use.* Used as an interface and information retrieval.
– *Delivery of sound.* Stereo.

This is an entirely audio interface which could be used by all users regardless of hardware or disabilities. The metaphor used is not based on the already successful visual models (i.e. Graphical User Interface). The metaphor used is based on a "bubble." Each "bubble" can contain a number of items, including other "bubbles." Each "bubble" has its own sound texture which can be heard while the user is in that particular "bubble" and for all child "bubbles." Dynamic audio can be triggered at specific times to alert the user to other events at any given time. The user can interact via a trackpad that allows the navigation of the "bubble" and five Braille-labeled keys on the numeric keypad that control the playback of audio. There were difficulties with over-load when there were more than three sounds at an equally loud level. More customisation and 3D sound was requested by testers.

6.3 Video Games

This model describes the function of sound in video games [8].

- *Cues to user.* This model uses a combination of audio and visual cues.
- *Type of sound.* This model uses auditory icons to describe the users situation in the game. It also includes background music to add to the overall effect, and to build up tension.
- *Number of simultaneous events.* Multiple events at any given time.
- *Areas of use.* User interaction
- *Delivery of sound.* This model users mainly 3D sound.

Sound plays an important part in role playing games. The sound tries to mimic real world sounds (if available) and provide clues to what is about to happen. It also adds to the realism of the game, keeping players "hooked." In some cases it is impossible to play without the sound, as key objects that are required by the gamer are signaled with sound only.

6.4 Hypothetical Model

This model describes a possible interface which could be used for both VI users and users of mobile computers.

- *Cues to user.* This model would use mainly audio cues, but would have some visual available.
- *Type of sound.* This model would use a combination of types of sound.
- *Number of simultaneous events.* Multiple events at any given time.
- *Areas of use.* Used for both user interaction and information retrieval.
- *Delivery of sound.* 3D sound with either headphones or speakers.

Using the audio cues to extend the monitor virtually, users can find the necessary information using a combination of audio and visual cues. Users could prioritise what is on the monitor. For example, a file being downloaded may not be immediately interesting, so that may be placed behind the user. A document that the user is reading is perhaps more important, so that could be in a more prominent position. An effective visualisation could be to think of a 3D information space, similar to that of a 3D web. The user is surrounded one layer of data at any given time. By moving to a different "layer" a different set of data becomes available. Layering of sounds could provide useful clues as to where the user is at a given time. These would probably have to be heard at all times.

Earcons have some training period, but as icons also have a similar training period, and users have overcome this, it is thought that this would not be an issue. The main benefit of earcons being that they are forever extensible, and once the initial training period is over, it should be relatively simple to use them efficiently.

Users are capable of decoding multiple streams of information at a given time. A limit would have to be placed on how many streams would be heard, or how similar the earcons were as over-loading could be an issue here. The use of 3D sound should help to reduce these possible errors.

3D sound allows sounds to be placed around the users head allowing a bigger virtual screen size to occur. Mono and stereo might be too limiting for this purpose.

7 Future Work

Our next steps include generating and implementing various models within the space to try to more clearly define the impacts of design choices in or model space. Promising models will be evaluated by users in our target groups (VI and mobile computer users). We hope to be able to empirically determine the impact of various model space decisions on users of our different target groups in order to inform our future audio interface designs.

8 Conclusion

At present the main focus for computer displays has been on creating them visually. As the use of the computer is developing, the reliance of graphics may have become a problem. In order to develop a successful metaphor for an audio user interface, it may be necessary to choose a different paradigm. Mapping a sound to an image may not be appropriate as visual and audio cues are treated differently by people. Graphics can remain on the screen for long periods of time, whereas in an audio environment they are temporal. We feel that this model space will provide us with a framework for testing out ideas for building effective audio interfaces.

References

1. Albers, M. C.: The Varese system, hybrid interface and satellite-ground control: Using auditory icons and sonfication in a complex, supervisory control system. *ICAD* (1994)
2. Brewster, S., Holguin C., Pirhonen A.: Gestural and Audio Metaphors as a Means of Control for Mobile Devices.*CHI Vol 4, Issue 1 pp 291 – 298* (2002)
3. Brewster, S. A., Raty, V. P., Kortekangas, A.: Representing complex hierarchies with earcons. *ERCIM Research Report 05/95R037, ERCIM*
4. Disappearing Computer: `http://www.disappearing-computer.net/mission.html`
5. Glasgow and West of Scotland Society for the Blind: Home page. `http://www.gwssb.demon.co.uk`
6. Helal A., Moore S.E., Ramachandran B.,: Drishti: An Integrated Navigation System for Visually Impaired and Disabled, *Proceedings of the 5th International Symposium on Wearable Computer*, October 2001, Zurich, Switzerland.
7. Heller, R.S, Martin, D., Haneef, N., Gievska-Krliu, S.: Using a Theoretical Multimedia Taxonomy Framework. *ACM Journal of Educational Resources in Computing, Vol.1, No.1, Spring 2001, Article 4, 22 pages*

8. Lombardi, V: `http://www.noisebetweenstations.com/personal/essays/audio_on_the_internet/SoundInInterface.html`
9. McGookin, D. K., Brewster, S.: DOLPHIN: The design and initial evaluation of multimodal focus and content. *ICAD* (2002)
10. Norman, D.A.:*The Design of Everyday Things* (MIT Press, London, 2000)
11. Royal National Institute for the Blind: Home page. `http://www.rnib.org.uk`
12. Sutcliffe, A.: *Multimedia and Virtual Reality – Designing Multisensory User Interfaces* (Lawrence Erlbaum Associates, Mahwah, NJ (2003))
13. Sawhney, N., Murphy, A.: Designing audio environments – not audio interfaces. *ACM Assets, the Second ACM/SIGCAPH Conf Assistive Technologies* (1996)
14. Sawhney, N., Murphy, A.: ESPACE 2: An experimental hyperaudio environment. *CHI Conf companion pp 105-106* (1996)
15. Shinn-Cunningham, B. G.: Spatial auditory displays. *International Encyclopedia of Ergonomics and Human Factors* W. Karwowski, Ed. London: Taylor and Franics Ltd (2000)
16. Walker, A., Brewster, S., McGookin, D., Ng, A.: Diary in the sky : A spatial audio display for a mobile calendar. *IHM-HCI* (2001)
17. Walker, A., Brewster, S.: Spatial audio in small screen device displays. *Personal Technologies 4(2) pp 144-54* (2000)
18. The vOICe: Wearable computing used by the blind `http://www.seeingwithsound.com`
19. Wearable Computing: `http://www.media.mit.edu/wearables/projects.html`
20. Web Accessibility Initiative: Home page. `http://www.w3.org/WAI/`

Developer Support in Open Hypermedia Systems: Towards a Hypermedia Service Discovery Mechanism

Nikos Karousos[1, 2] and Ippokratis Pandis[3]

[1] Research Academic Computer Technology Institute
16500, Rion, Greece
karousos@cti.gr
[2] Dept. of Computer Engineering & Informatics, University of Patras
26500, Rion, Greece
[3] Athens Information Technology (AIT)
19002, Peania, Athens, Greece
ipan@ait.gr

Abstract. This paper argues that the open hypermedia research community should focus on the developer support issue. An enabling step to this target is the provision of an effective and easy-to-use hypermedia service discovery mechanism. The inexistence of such kind of mechanism, for finding and using hypermedia services, amongst the Open Hypermedia Systems (OHSs) is one of the main reasons for their narrow publicity and usage. Aiming to the improvement of the third party (hypermedia-unaware) developer support, the hypermedia service discovery mechanism will boost the OHS usage by providing a standard platform and a set of tools in order to enable the enhancement of third party applications with hypermedia functionality.

1 Introduction

During the last years, hypermedia systems researchers have focused on the implementation of service-oriented component-based open hypermedia systems [21]. The Open Hypermedia Systems Working Group (OHSWG) [14] through the Open Hypermedia Protocol (OHP) [15] tries to establish the basis, in order the hypermedia systems developers to be able to build open and interoperable systems. It is obvious that researchers target to a situation where any hypermedia system could co-operate and work with any other hypermedia system as well as could easily be integrated with any other third party application; however, the usage of the OHSs is still narrow and fails to reach a critical mass of users [12]. In addition, many people are unaware of the existence of the whole category of hypermedia systems that could fulfill their needs and expectations for data management, manipulation, structure and linking.

One of the reasons that led to the aforementioned situation is that despite the efforts for achieving open-ness in OHSs, those efforts were ad-hoc solutions for each system and none had emphasized on the (third party) developer support issue. The OHS community has not targeted effectively, yet, in the establishment of common methodologies and frameworks that could help any developer to insert hypermedia functionality into his application. Because of the inexistence of such infrastructures,

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 89–99, 2004.

OHSs are practically used only by users inside the small OHS community. Thus, the need for facilitating the insertion of hypermedia functionality into third party applications, in a common way, is demanding. In this paper we are focusing on one of the most important issues that rise in the developer support field: the need for a hypermedia service discovery mechanism.

The first step for a developer, who wants to add hypermedia functionality to his application, is to search for an available OHS, which provides hypermedia services and satisfies his requirements. Until now, in the area of hypermedia systems, a standardized methodology for discovering capable hypermedia services does not exist. Moreover, there is not either an inchoate record of the existing hypermedia systems. Developers usually obtain information about hypermedia systems out-of-band, by other developers, or try to re-use already used and known systems. The difficulty of discovering the capable system is an obstacle for many developers in order to add hypermedia functionality into their applications.

The analysis of the OHSs characteristics from a discovery mechanism point of view, leads to the conclusion that OHSs should be outfitted with a new feature. This feature is the hypermedia server ability to describe itself. In that way, hypermedia systems are equipped with *introspection capabilities*. This self-description capability can be achieved, when the system is aware of its API and the semantic of its operations, namely, to be completely aware of its provided hypermedia services. Then, it will be able to answer questions like: “who are you, what services can you provide and how can I have access on this specific operation?” The developers of hypermedia systems can rely on the introspection capabilities of these systems and built more sophisticated tools, which will aid the hypermedia-unaware developer. Furthermore, it will be much easier to accomplish the target for open and interoperable hypermedia systems if these systems are equipped with introspection capabilities.

The remainder of this paper is structured as follows. Firstly, the concept of server discovery is presented, followed by a section dedicated to the hypermedia service discovery. We present the currently adopted situation and we argue over the need of a hypermedia service discovery mechanism. In section 4, we analyze the main subjects towards a hypermedia service discovery mechanism, while in section 5 we present some extra features and tools that can be added into a hypermedia service discovery mechanism. Finally, we outline directions for future work and present our conclusions.

2 Service Discovery

In networking the word “service” means any facility that is available on the network [16]. Thus, when we refer to “service discovery” we mean the procedure of locating (inside or outside the network) the sources where information about servers can be found, querying these sources for available servers and matching the providing services of these servers with the user needs. Service discovery is completed when the requester is provided with the required server access information.

The need for server discovery is uprising by many distinct user groups. Namely, when multiple servers with the approximate provided services exist, or a user uses a specific server and wants to pop to another for optimized performance, the need for server discovery arises. Currently known groups of users who apply server discover-ies are the on-line game players, the users of file sharing applications, like the Napster-like applications, the users of chat servers, and indiscriminately all the users of on-line multi-user applications.

How users obtain information about servers? Except from the usual way of searching through a search engine (Google, etc…) for some keywords, or the out-of-band information obtention i.e. from another user, there are some specific purpose systems. At this moment, discovery systems based on different architectures, like the centralized server discovery systems and the distributed and/or peer-to-peer server discovery systems, can be found.

Server discovery is a very domain-specific issue. It is a procedure heavily depended on the specific characteristics of the server, which will be requested by the users. For example, the requirements and the concerns for a developer who creates a server discovery system for popular servers with an a-priory defined set of provided services are completely different with the ones who concern a developer of a server discovery system for a very distinguishable type of servers, like the hypermedia servers. Hypermedia systems are a class of systems with a variety of special characteristics. Thus, in the following section we ponder over the problem of server discovery in hypermedia systems.

3 Service Discovery in Hypermedia Systems

3.1 The Need of Service Discovery in OHSs

In hypermedia world, hypermedia service can be defined as a facility, which provides the users with the ability to invoke operations of hypermedia content. These operations belong to a larger infrastructure, called hypermedia services provider or hypermedia server. Hypermedia services can emerge directly by a single hypermedia structure server operation or by a subset of the operations supported by a structure server in an OHS. For example, the navigational link management service in an OHS could be provided as a service on its own. On the other hand, more complex services such as versioning, storage, transaction and searching for hypermedia documents could be published as independent hypermedia services to developers.

The service discovery in OHSs has not acquired, yet, its deserved attention. It is noticed that in hypermedia systems there are neither any hypermedia service discovery mechanism nor any detailed record of those systems. Open Hypermedia Systems will benefit from the adoption and implementation of a hypermedia service discovery system. First of all, as it is pointed out [5], one of the major issues that afflict the OHSs is that despite the research and the implementations that have been accomplished in this area, hypermedia systems remain unknown outside the borders of the small OHS community. The difficulty in discovering desired hypermedia servers often deters simple users and third party developers from using the OHSs. From another point of view, the researchers in the OHSs face difficulties in accessing approximate systems or get incomplete information about them. Only through published papers, personal communication and OHSs' presentations they can be informed about new features and innovating characteristics of the implemented hypermedia systems. A service discovery mechanism can act as an access point for the researches to other hypermedia systems, and help them explore their functionality.

The design of Component Based Open Hypermedia Systems (CB-OHSs), where multiple structure servers exist, strengthens the need for a server discovery system. The potential users of such a mechanism may not be only the simple users or the third party developers, but even the hypermedia systems themselves. Hence, an automate mechanism could be created for the CB-OHSs in order to make use of this hypermedia service discovery mechanism and dynamically find and use the most appropriate hypermedia server. Such architecture results in a complete service-oriented approach in hypermedia systems.

A real-life scenario, which describes the current situation, is presented below. When a developer intends to insert hypermedia functionality to his application, he usually follows the next steps: (a) Searches for the appropriate hypermedia server through the network. The hypermedia server location can be obtained out-of-band, for example from another developer who had used the same hypermedia server, or from own prior experience. (b) Understands the server interface and the way the server operates. (c) Implements the required communication level into the application in order to communicate with the server. (d) Finally, encapsulates the adopted service as a set of functions added into his application. The aforementioned adopted procedure is unacceptable, from a third party developer perspective, and it is a challenge for the hypermedia community to address new methodologies for better developer support.

3.2 Related Work

Many researchers have mentioned the need of a service discovery feature into the area of OHSs. Towards this, implementations like the Server Information Management (SIM), which collects server name and location information from servers and distributes it to clients [11] and the notion of Hypermedia Resource Descriptors (HRD) that can describe both documents and servers [17] have been proposed.

On the other hand, common architectures like Web Services and CORBA try to address the arising problem of service discovery by using a global registry mechanism (Universal Description, Discovery and Integration of Web Services – (UDDI) [18] and an internal naming and discovery mechanism respectively. The service discovery issue in Web Services is subject to great research. New technologies like the so-called Web Service Discovery Architecture [4] and the Open Grid Services Architecture (OGSA) [13] provide different approaches from the classic UDDI for the service discovery.

The Open Hypermedia Protocol (OHP) was designed to address interoperability problems between different hypermedia servers. However, it contained aspects of functionality, available to enrich the provided services of hypermedia servers. Most notably, OHP can help a client to discover all the services of a specified hypermedia server [10]. Dynamic service discovery via OHP is a notable tool, but its main disadvantage is that it provides no help for the client to discover the hypermedia server. The work of OHSWG in this area can be proven very helpful. In a hypothetical hypermedia server explorer the user will be able to navigate through hypermedia servers, making the discovery procedure even more effective.

4 Analysis of a Hypermedia Service Discovery Mechanism

An enabling step towards the implementation of a discovery mechanism for hypermedia services is the analysis of the most important issues of a proposed hypermedia service discovery mechanism. This task is still in progress. The first outcomes are presented below.

4.1 Hypermedia Service Discovery Architecture

There are two different approaches for the architecture of the service discovery mechanisms: the centralized and the distributed service discovery architectures.

4.1.1 Centralized Service Discovery

In this architecture the following procedure is applied: When a server starts operating, it registers to a hypermedia service directory server. A potential user queries the service directory, which has a well-known location, for desired services. The service directory responds with a list of services in which the user may find the desired service.

The centralized technology is used for several purposes, for example for discovering *first person shooter* game servers [3,19]. However, there are some drawbacks in this architecture. In detail, when the service directory server goes down the whole service stops. A server can register to many server directories or to the same server directory many times, thus, the directory service has to apply many checks before inserting an incoming server into the directory and to perform periodical maintenance functions.

A representative paradigm of a centralized service discovery mechanism is the UDDI project that is used by the Web Service community.

4.1.2 Distributed Service Discovery

The distributed server discovery systems consist of a set of server directories that interoperate with each other by passing queries in order to discover the information about appropriate servers from a distributed server directory system. Furthermore, a new approach of peer-to-peer (P2P) server discovery systems, based on a full P2P architecture has increased the information distribution.

The distributed architecture resolves some of the already mentioned problems of centralized systems. However, it breeds some other issues, like complexity, security, latency of response time, QoS, etc. Another problem is the maintenance of information distributed from a server directory to other directories. These issues are the objectives of the current research domain in distributed architectures resulting some interesting approaches [23,2].

4.1.3 Choosing the Appropriate Hypermedia Service Discovery Architecture

As it has been mentioned in the previous section, a service discovery mechanism is heavily dependent on the specific characteristics of the servers it works with. So, in order to choose the most suitable architecture for a discovery mechanism in hypermedia systems, we consider the specific characteristics of those systems. The Open Hypermedia Systems do not consist of a large set of hypermedia servers (or structural servers) that provide their services outside of their environment. In addition, the fact that no hypermedia service discovery mechanism has yet been adopted makes us strive for the design of a simple discovery mechanism.

The most reasonable first step is to build a centralized Hypermedia Service Discovery mechanism that can fulfill all the required needs of the current hypermedia systems. However this mechanism has to be extendable in order to be able to support a future transformation to a distributed service discovery system.

4.2 OHSs Introspection Capabilities

The key feature, in which the discovery mechanism is based on, is the introspection capabilities of a hypermedia system. We define as introspection capabilities of a hypermedia system, the awareness of all of its provided hypermedia services. A hypermedia system should be aware of all provided hypermedia services and how those can be invoked.

A first step for addressing introspection is the hypermedia servers' self-described ability. For this reason, we believe that a Hypermedia Service Description Language (HSDL) should be defined. This language can be based on emerging standardized markup languages, such as the Web Service Description Language (WSDL) [20] and the Resource Description Framework (RDF) [7], or it can be a new XML-based ad-hoc language. The basic concept of the Semantic Web [1] is that the data may not be only machine-readable and human understandable, but also machine understandable [8]. Therefore, we believe that a hypermedia service description language can only have an XML format.

This language should be used to describe a hypermedia service so as to be able to provide all the necessary information in order to communicate and interoperate with external applications. More specifically, the HSDL should be structured in a standard way and should include the following information:

1. **Service General Information.** That includes information about the service name, the service provider, the service scope and finally structural specific information such as the hypermedia domain (e.g. spatial) etc.
2. **Service Locating and Accessing Information.** It includes the host and port (or other location information) and the required communication protocol(s) between client and hypermedia server.
3. **Service Interface Information.** In the interface information section the set of the available operations that the service supports is described analytically. These operations are defined as functions that can be called remotely from a client and can return a result. Furthermore the set of the necessary data types that is passed or returned as function parameters is defined.
4. **Service Behavior Information.** This type of information refers to the behavior of the hypermedia server, after each request, and the dependences between the server operations.
5. **Service Comments.** The hypermedia service commentary is useful when developers read the HSDL while trying to understand the way the server operates.

Finally, the HSDL should be extendable and open in order to be able to augment new language characteristics and specifications for supporting specific servers or servers of new hypermedia domains. In addition the description of the naming methodology that every OHS use has to be also supported of the HSDL.

The current research on both the Web Services and the WSDL can be the first step for the creation of the HSDL. The extension and the alteration of the WSDL could be a very interesting task, aiming to the definition of the first version of the HSDL.

4.3 Hypermedia Service Registry

When a hypermedia service is ready to operate, a service register invocation has to be followed: a) a registry record with some required fields that describe the service must be filled up and b) either the system administrator or the OHS by itself has to register the service by passing a request with the registry record to the service directory.

A service registry should consist of a set of fields that are able to describe the service and can be queried from potential users who aim to discover an appropriate service. A list of the most important required fields in the registry record is presented below.

Table 1. List of the most important fields in the registry record.

Registry Fields
Owner Information
Owner Name, Address, URL, etc.
OHS Information
OHS Name, OHS Description
OHS Access Point
Domain Information
Domain Type
Service Information
Service Name, Service Description
Required Communication Protocol
Service Status
Service Location
HSDL Location
Registry Information
Registry Unique Name / ID
Register Date, Duration, Description, etc.

The specification of the data that will be provided in the registry has not been fully determined yet. For each service some extra metadata concerning the service authentication, the license policy etc. should be provided. These metadata can be clustered into semantic chapters e.g. metadata for the technical issues of the service, metadata for the economic issues of the service, metadata for the usage of the service, metadata for semantic issues of the service.

A very important issue that must be addressed for the hypermedia service discovery mechanism is the periodic checking of the advertised hypermedia servers' status. In case that an already registered hypermedia service does not work the corresponding registry record should not be available to the public.

4.4 Hypermedia Service Query Language

A potential developer, who wants to use a hypermedia service into his application, must be able to search and locate services using a service query language. This language should be based on a common used communication protocol and must be able to be used by both humans and applications. Then, the hypermedia directory system will be a service by itself that will respond to searching requests in a standard common protocol. The query result could be a list of hypermedia service registries that matches all the requested criteria.

For example, when a requester queries the service directory for a navigational link server that communicates over OHP-Nav [9], the service directory should respond with a list of all registries that describe navigational links servers which are using OHP-Nav. It is preferred this list to be in XML format.

Furthermore, in order for the service to result the optimum appropriate hypermedia services for the requests, a set of effective ranking algorithms should be applied to the matchmaking procedure of the mechanism.

4.5 Hypermedia Service Discovery Usability

Aiming to the growing usage of the OHSs, the discovery mechanism has to be available and easy to use by both humans and computers.

One or more hypermedia directory web interfaces should be created. In this way, it will be possible for the hypermedia community to inform the whole web community about their provided services. In addition, the web service directory interface should operate as a visual representation tool, which classifies the services into categories and provides them to the public through tree-views with multiple perspectives. For a user of this interface, querying for a service will be an easy task.

An optional requirement is the existence of an available demo for each published service. With this way the user will be able to see exactly what request a published hypermedia server can serve and to test if a selected service really does what he expects.

5 Add-on Features of the Hypermedia Service Discovery Mechanism

The service discovery mechanism does not require any alteration of the existing architectures and methodologies. Nevertheless, there is a set of issues that concerns the OHS designers and should be addressed in order to provide some extra benefits to the proposed mechanism.

5.1 Hypermedia Services Exploring Tools

Giving an OHS access point to the developers, an application that allows the exploration of available services of a particular hypermedia system should be available. In addition, a more complicated but with global scope tool, which allows the navigation through all the available services of the hypermedia systems should be provided. In this way, the global view will contribute understandability of the entire hypermedia system.

5.2 Communication Protocol Awareness

The developer should not worry about the communication protocol that has to be used in order to communicate with a specific hypermedia server. Therefore, approaches that utilize widely accepted communication infrastructures such as the Simple Object Access Protocol (SOAP) should be adopted, or the ability of dynamically locating and loading protocols – at run time -should be investigated. Such dynamic behavior can already be witnessed in applications such as browsers and video players that automatically download plug-ins.

As a result, when the discovery mechanism returns a list of matched registry records, for each selected service that uses a well-known communication protocol, the required protocol access information will also be provided.

5.3 Automatic Client Skeleton Creation

Tools for automatic generation of the client communication layer should be developed. Such kind of tools should be available for a wide range of devices and platforms.

Some significant propositions have been made in this issue. An interesting approach is presented in the *Construct* environment [22]. According to Construct, the developer is able to use tools for the automatic creation of skeleton code from a UML diagram or an IDL specification of a service. However, not all of the prospects have been explored so far. For example, if a standardized hypermedia service description language was available then some more sophisticated and effective automatic client creation tools could be created.

6 Future Work

Developer support in OHSs has been the main target of our work for the past few years [5,6]. During our research we focused on the need for creating a hypermedia service discovery mechanism. Studying both the Web Service discovery architecture and the current status of the OHSs, we strongly believe that the creation of a centralized hypermedia service discovery system is the only feasible task. Despite the fact that research on distributed discovery issues has reached a considerable level, the implementation of such distributed discovery systems in OHS community is unattainable for the time being. The reason lies to the huge complexity of the OHSs and the narrow research on hypermedia discovery issues.

Below are presented the future steps that must be followed in order to achieve the target goal:

- *Definition of the service registry record.* The fields' definition of the service registry record will take place in two phases: Initially, a first version of the registry record should be created. This record will include service locating and describing information. Secondly the location of the HSDL service description will be appended in the record. This presumes that the HSDL should already be defined.

- *Implementation of the first registry directory.* It includes the implementation of a service directory, which will serve requests for service registration and searching. The definition of a simple communication protocol for the directory should also be made here. Applying Web Service in association with Simple Object Access Protocol (SOAP) as a first step seems considerably interesting.
- *Web – based directory interface creation.* In order to create the first on-line OHSs' service directory our next step is to implement a web interface that will publish the registry directory to the web.
- *Definition of the HSDL.* Defining the HSDL is probably the most difficult task of all and for this reason we believe that it should be made step by step, by extending it each time in order to cover initially all the descriptions of the hypermedia services for each one of the already known domains (navigational, taxonomic, spatial etc.).
- *Creation of the add-on tools.* After the above research, the implementation of graphical representation tools for the available hypermedia services per OHSs, both for use inside every OHS and for integrated use in the discovery mechanism environment, will be possible.

Although the above tasks require a huge effort in order to be accomplished, we strongly believe that the necessity of such a discovery infrastructure is imperious. Many OHSs researchers should co-operate in order to define the appropriate common protocols and requirements in a proper and effective way.

7 Conclusions

Service discovery in hypermedia systems is not a supported feature yet. The implementation of a hypermedia service discovery mechanism will increase the OHS usage and highlight the advantages of the provided hypermedia services. This task will also be a great step to achieve the growth of the OHS popularity.

Considering the existence of the current architectures in OHSs and the work of OHSWG (OHP) we suggest the development of a simplified hypermedia service mechanism. Furthermore, we encourage the hypermedia community to participate in the standardizing of both the protocols and the requirements towards an effective and fully-working hypermedia service discovery mechanism.

Acknowledgments

The authors would like to thank Manolis Tzagkarakis for his insightful comments and advices.

References

1. Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., and Secret, A. (1994). The World-Wide Web. Communications of the ACM, 37(8), pp. 76-82.
2. Gibbins, N. and Hall, W. (2001). Scalability Issues for Query Routing Service Discovery. In Proceedings Proceedings of the Second Workshop on Infrastructure for Agents, MAS and Scalable MAS, pages 209-217.

3. Henderson, T. (2002). Observations on game server discovery mechanisms. In Proceedings of NetGames 2002, (Braunschweig, Germany), Apr. 16-17 2002.
4. Hoschek,W. (2002). The Web Service Discovery Architecture. Proc. of the Int'l. IEEE/ACM Supercomputing Conference (SC 2002), Baltimore, USA, November 2002. IEEE Computer Society Press.
5. Karousos, N., Pandis, I., Reich, S., Tzagarakis, M. (2003). Offering Open Hypermedia Services to the WWW: A Step-by-Step Approach for the Developers. In Twelfth International World Wide Web Conference WWW2003, (Budapest, Hungary), pp. 482-489.
6. Karousos, N., Tzagarakis, M., and Pandis, I. (2003). Increasing the Usage of Open Hypermedia Systems: A Developer-side approach. In Proceedings of the ACM Hypertext 2003 Conference, (Nottingham, England).
7. Lassila, O., and Swick, R. R. (1999). Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, 1999. At URL: <http://www.w3.org/TR/REC-rdf-syntax/>
8. Li, L., and Horrocks, I. (2003). A software framework for matchmaking based on semantic web technology. In Proc. of the Twelfth International World Wide Web Conference (WWW 2003), pages 331-339, 2003.
9. Millard, D. E., Reich, S., Davis, H. C. (1998). Reworking OHP: the Road to OHP-Nav. In 4th Workshop on Open Hypermedia Systems (OHSs) at Hypertext' 98, pp.48-53.
10. Millard, D. E., Reich, S., Davis, H. C. (1999). Dynamic Service Discovery and Invocation, In Proceedings of the 5 th Workshop on Open Hypermedia Systems, TR CS-99-01, Aalborg University Esbjerg, DK, pp. 38-42, 1999.
11. Nürnberg, P. J., and Legget, J., J. (1997). A Vision for Open Hypermedia Systems. Journal of Digital Information (JoDI). Special Issue on Open Hypermedia Systems 1, 2.
12. Nürnberg, P. J., and schraefel, m. c. (2002). Relationships Among Structural Computing and Other Fields. JNCA Special Issue on Structural Computing, 2002.
13. Open Grid Services Architecture Working Group – OGSA WG site. At URL: <http://www.ggf.org/ogsa-wg/>.
14. Open Hypermedia Systems Working Group – OHSWG WWW site. At URL: <http://www.cs.aue.auc.dk/ohswg>.
15. Reich, S., Wiil, U. K., Nürnberg, P. J., Davis, H. C., Gronbaek, K., Anderson, K. M., Millard, D. E., and Haake, J. M. (2000). Addressing interoperability in open hypermedia: The design of the open hypermedia protocol. Special issue on open hypermedia. The New Review of Hypermedia and Multimedia, 5, pp. 207-248.
16. Tanenbaum, A.S. Computer Networks, 3 ed. Prentice-Hall International, Englewood Cliffs, New Jersey, 1996.
17. Tzagarakis, M., Karousos, N., Christodoulakis, D. and Reich, S., Naming as a fundamental concept of open hypermedia systems (2000). In Proceedings of the 11th ACM Hypertext Conference, pp. 103–112, (San Antonio, TX, USA).
18. Universal Description, Discovery and Integration of Web Services (UDDI). At URL: <http://www.uddi.org>.
19. Valve Software. Half-Life. At URL: <http://sierrastudios.com/games/half-life/.>
20. W3C Web Services Description Language (WSDL). At URL: http://www.w3.org/tr/WSDL
21. Wiil, K. U., Hicks, D. L., and Nürnberg, P. J. (2001). Multiple Open Services: A New Approach to Service Provision in Open Hypermedia Systems. In proceedings of the 2001 ACM Conference on Hypertext and Hypermedia, (Aarhus, Denmark), p. 83-92.
22. Wiil, U. K., Nürnberg, P. J., Hicks, D. L., and Reich, S. (2000). A development Environment for Building Component-Based Open Hypermedia System. In Proceedings of the 11th ACM Hypertext Conference, (San Antonio, TX, USA), pp. 266-267.
23. Xu, D., Nahrstedt, K., Wichadakul, D. (2001). QoS-Aware Discovery of Wide-Area Distributed Services, Proceedings of IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGrid 2001), Brisbane, Australia, May 2001.

A Structural Computing Model for Dynamic Service-Based Systems

Peter King[1], Marc Nanard[2], Jocelyne Nanard[2], and Gustavo Rossi[3]

[1] Department of Computer Science, University of Manitoba
Winnipeg, MB, R3T 2N2 Canada
prking@cs.Umanitoba.ca

[2] LIRMM, CNRS/Univ. Montpellier, 161 rue Ada
34392 Montpellier cedex 5, France
{mnanard,jnanard}@lirmm.fr

[3] LIFIA, Fac. Cs. Exactas- Universidad Nacional de La Plata
(1900) La Plata, Buenos Aires, Argentina
gustavo@sol.info.unlp.edu.ar

Abstract. Traditional techniques for Programming in the Large, especially Object-Oriented approaches, have been used for a considerable time and with great success in the implementation of service-based information systems. However, the systems for which these techniques have been used are *static*, in that the services and the data available to users are fixed by the system, with a strict separation between system and user. Our interest lies in currently emerging *dynamic* systems, where both the data and the services available to users are freely extensible by the users and the strict distinction between system and user no longer exists. We describe why traditional object-oriented approaches are not suitable for modelling such dynamic systems. We discuss a new architectural model, the *Information Unit Hypermedia Model*, *IUHM*, which we have designed for modelling and implementing such dynamic systems. IUHM is based upon the application of structural computing to a hypermedia-like structure, which thereby operates as a service-based architecture. We discuss the details of this model, and illustrate its features by describing some aspects of a large-scale system, built using this architecture.

1 Introduction

An important current trend in system design and development is the consideration of dynamic systems, particularly dynamic service-based systems. In this paper, we present a new architectural model for modelling and implementing such systems. Specifically, we are interested in dynamic systems where users of the system are free to create new data and new data types together with new services for the interrogation and manipulation of such data. Furthermore, the addition of new types and services must not require any explicit changes, upgrades or reorganization in existing parts of

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 100–118, 2004.

the system; indeed existing types and services should be able to make automatic use of such additional components[1].

Indeed, the existence of these two forms of user extension, new data and new services, serves to characterize the class of system of interest to us. The dynamic nature of both data and functionality within such a system has very considerable effect on the viability of implementation approaches. Whereas traditional techniques for programming in the large, particularly object-oriented approaches, provide substantial support for low- and medium-scale programming, these approaches do not lend themselves as readily to a number of the specific issues arising in such large-scale dynamic systems. The work we describe herein shows how *structural computing* [23] techniques, based on graphical descriptions of relationships between system components, are particularly applicable to the management of large-scale extensibility and tailorability. These techniques provide a means to describe formally the structure of the system and to depict properties of items within the system. At the same time, properties similar to those that have proved useful in object-oriented approaches, such as inheritance, polymorphism, and delegation, are readily described by these structural techniques. Thus, our overall approach to implementing such dynamic systems is a combined one, in which we use traditional object-oriented programming for programming the individual system components "in the small", and use the structural programming approach to be described in this article to provide an implementation model for the other aspects of the system. The systems which we are considering in this article are not *auto-adaptive*, but depend exclusively on user interactions for their extensibility. The systems we have in mind are large and may involve many users.

The architectural model which we describe in this paper is a unified, reflexive one, in which all entities (data, metadata, service, ontology, etc.) are represented in a uniform fashion; each entity is encapsulated as an *Information Unit* (IU), and relationships between entities are denoted by an explicit graph structure, built as a linked network of IUs. This linked network is analogous to a hypermedia structure. Manipulations take the form of the application of structural computing to this hypermedia structure, which thereby operates as a service-based architecture. This approach enables us to apply well-known meta-level programming techniques in order to reason on the system structure (the meta-level), just as we reason on the system data (the base level). Furthermore, the IU maintains a distinction between *structure* and *semantics* in the manipulation of an entity. As we explain in what follows, each IU contains a number of links (pointers to other IUs) and in particular, an IU has a *type* link and a *role* link, which are both dynamic and which correspond, respectively, to the structure and to the semantics of the entity represented in the information unit in question.

The approach to be discussed in this paper describes, naturally, a conceptual rather than a physical architecture, that is to say the model says nothing about the physical locations of the actual software elements. However, the approach has been used in practice, in the implementation of the OPALES system, designed and implemented for INA, (the National Institute for Audiovisual Archives in Paris). We will use

[1] We use the term tailorability to describe this aspect of the system.

OPALES to illustrate several of the ideas in this paper. OPALES provides a portal to a set of diverse open digital library services. OPALES is a typical instance of the class of large-scale dynamic system of interest to us, comprising as it does some 80,000 lines of Java code, and the same amount of C++ code [4], [20], and being specifically designed for the cooperative manipulation of shared multimedia documents among multiple users and user-groups. In particular, such documents may be enriched by multiple *annotation* and *indexation* structures through private and public workspaces. The techniques we describe are intended for dynamic systems such as OPALES; however, they also work well where there is a limited degree of extensibility – where, for example, new upwards compatible, functionality may be added in a system-controlled fashion.

The remainder of this paper is organized as follows. In section 2 we further describe the general context of our work, and we discuss the design rationale for the architecture to be described. Section 3 briefly describes *IUHM* the *Information Unit Hypermedia Model*, our infrastructure for modelling dynamic systems. Subsequent sections go into further detail of how this infrastructure meets the requirements described in section 2. Thus section 4 discusses how the IUHM model may be used for modelling and implementing a dynamic system, We introduce and describe in detail the concepts of *role* and *type*, and discuss how these concepts lead to a simple resolution of the question of *interoperability* between system modules. Section 4 also discusses the notion of *reflexivity*, and describes in detail the *information unit* which is at the core of our infrastructure. Section 4 also explains the dynamic mechanism used to run the *pattern matching* in the implementation of our model. In section 5, we briefly discuss a number of important aspects of related work, and section 6 concludes.

2 General Context

In this section we consider the general context on which our work is based. In particular we discuss the essential differences between what we term *static* and *dynamic* systems, for it is these differences which provide the motivation for the architectural design which we will discuss in later sections.

2.1 Static Systems – Description

By the term *static* system we refer to a system in which there is a fixed number of pre-defined services available to users of the system, that is the set of allowable user operations and the available data (and data types) are both pre-determined. In such a system, there is a clear distinction between the system developers and the system users. Users are permitted to access particular services which access and/or modify data in well-defined ways. The use of such systems is widespread, and such systems include banking systems, travel reservation systems, university on-line registration systems, and so forth. In such systems data security is of primary importance, and in addition, therefore, to careful user authentication, such systems expressly exclude any facility whereby a general user could create a new service, since such a service could access data in a non-authorized fashion.

It should be noted that the distinction between user and developer is more usually a distinction between classes or levels of user: thus one may in particular have a class of super-user, who have responsibility for system maintenance, including updating existing services and the creation of new services. Generally speaking, making such new services available to other general users takes the form of a new upwards-compatible system release.

2.2 Static Systems – Implementation

Such static systems lend themselves well to traditional *programming in the large* implementation approaches [5]. In particular, the services provided in such a system are fixed, and are usually classified into a number of distinct categories. Within each category, the available services are to a large extent hierarchical in nature, and thus the traditional object-oriented approach is an appropriate one.

2.3 Dynamic Systems – Description

By way of contrast, a *dynamic* service-based system is one in which developers are free to add new classes of data and new services to access such data at any time, such as in [9], without the need to suspend, to reorganize or to re-release an entire system. Moreover, end-users may also restructure the system architecture to tailor the system to their own needs, and may create new services as compositions of existing services; such user level adaptation may imply quite deep component restructuring. Indeed, user operations of this sort are the intent of a dynamic system. Such large-scale changes imply that provision for automatic reorganization of computing within the system is one of the primary requirements of the system architecture.

Dynamic systems are less common, and thus we provide some examples of user activity by describing some typical user-induced system extensions. We take our illustrations from the OPALES digital video library system. We give a more detailed description of how our model handles such systems in a later section.

2.3.1 Developer-Based Extension

OPALES provides several tools for indexing and retrieving data, including queries based on descriptors, on keywords, on text similarity, or conceptual graphs and so on. Let us suppose we also want to support the Conceptual Vectors [13] querying technique. The data type and its associated set of tools are first implemented in the small in Java, say, in an IUHM [21] compliant manner. This new type of data and tools are then added into OPALES simply by setting, in a formal manner, the relationships between this new data-type and existing types by means of links between IUs. In this manner, any existing service concerned with indexing tools or data has automatic access to the new components. For example, the general querying service, which has the ability to combine expressions given according to different formalisms so as to build queries, will then automatically support also the Conceptual Vector technique, without any further updating. Similarly, annotating and indexing tools will gain access to the associated new editor.

2.3.2 End-User Based Structuring

OPALES provides private and sharable workspaces to its users. A user can build and organize a workspace and dedicate it to a specific *workgroup*. For instance, suppose that the user is an ethnologist, and wishes to annotate a video as a member of this interest group. The role of annotations created on the behalf on a given group is called the *viewpoint* of this group. The user may define work-rules within the workspace and restrict the data created by users of this group to be handled only by a set of specific tools, according to specific rules. Such a restriction is achieved by simply specifying the appropriate relationships between the data-type defining the interest group in question and the other items of the system. Any service in the system then automatically, configures accordingly, thus providing for the extensions, restriction and reconfigurations appropriate to this user group.

These examples taken from the OPALES system are by no means exclusive, and are cited to illustrate the types of user service which might be found in a dynamic system and which our architecture is designed to support.

2.4 Dynamic Systems – Implementation

The evolutionary nature of service and data creation within such a dynamic system means that a traditional object-oriented approach will not be sufficient, for a number of reasons.

- The creation of new services by users is unpredictable and, in general, non-hierarchical. Thus, it is not possible to pre-define a set of classes, much less a class hierarchy, which can model such services.
- Moreover, in the case of a system which is dynamically evolving and in which dynamic evolution is of the essence, the functional view which one has of each individual component does not lead to an overall view of the system as a whole.
- Indeed, users may create quite diverse services, so that the very concept of "the system as a whole" as a discernible item is lacking.
- Further, the creation of new services must be implemented in an evolutionary manner, without the need to modify the remainder of the system, or the need to resort to discrete versioning.

In the following sections we discuss an architectural model designed to implement such dynamic systems, and therefore to meet the various points just listed.

3 The Information Unit Hypermedia Model

The Information Unit Hypermedia Model, IUHM, is fully described in [21], and here we briefly review the model to the extent needed for the purposes of this paper. We observe that in IUHM, system construction corresponds to the specification of a network, and that the primary idea of IUHM is to provide a graph-based description of relationships between *Information Units* which encapsulate any entity (data, metadata, services), so that structural computing techniques can be applied. The type-based hypermedia structure we have chosen for IUHM induces a generalized typing mechanism on objects which is far richer than the classical class inheritance graph of object-oriented classes. Changing a link in the structure has an impact on the actual type of any object. As we discuss in the following section, the originality of the IUHM Model is that each tool may set its own type matching rules, enabling a late binding which relies on the structure of the actual IU graph.

3.1 Design Rationale

The important notion of *type matching* in the context of programming in the small is well-known. Notions such as classes, polymorphism, inheritance and so on have proven their efficacy in object-oriented programming. Programming in the large with dynamic binding of services and data, and composition of services requires a distinct paradigm which is suited to the specification of the rules which apply when data are assigned to services and the specification of how services cooperate. This section informally introduces the fundamental notions on which IUHM is based; these notions are developed in detail in subsequent sections of this paper.

We introduce two new notions: *surroundings* and *affinity*.

- The *surroundings* of an item characterizes the relationships between that item and the others in the system. In contrast to data types, surroundings is not local to an item but is affected by structural changes which occur around an item. Surroundings characterizes not simply the data but all the relationships between one IU and others, and thus the surroundings of an IU potentially includes other IUs which are quite distant in the graph structure. The notion of surroundings is significant in that it provides a means to trigger or inhibit actions on data from other arbitrary items without knowledge of concerned items in question.
- As in the social world, *affinity* depends upon surroundings. The affinity of a service refers to the kind of surroundings that must have the items it is willing to process. A service can define its affinity, and the affinity rules determine which properties of the surroundings are appropriate in a particular instance. The notion of affinity thus introduces a generalization of type matching, which encompasses but which is far richer and has a higher expressive power than the type matching to be found in programming languages.

3.2 Information Units

IUHM has its origins in a hypertext model and is the result of a long evolution and enrichment of our work on typed links hypertext systems [19]. IUHM represents information in a form which resembles a hypertext with typed links and typed nodes. These nodes *encapsulate* data within a surroundings, namely the hypertext network itself, on which structural computing may be performed to compute actual affinities. An original aspect of IUHM, and one which demonstrates a fundamental difference from object-oriented approaches, is that services and data are fully unified, that is all nodes encapsulate data. The data in question may in particular be code, depending only on its surroundings. We refer to this node as an *Information Unit, IU*. An IU is connected to other IUs by *links*, which express different types of properties of the surroundings.

IUHM introduces the notion of *role* and makes an explicit distinction between the notion of role and the notion of *type*. The *type* provides information needed to handle the data at low level, whereas *roles* are the high-level actions in which that data is involved. Provided that a given set of types share a given interface (say they inherit from a given type), several IUs of distinct types belonging to this set may share the same role; that is, the same high level actions are possible regardless of the underlying low structures. In the IUHM model therefore, each information unit, IU, has two required links, the *type* and the *role*. Thus, every IU is related to at least two other IUs, which represent its type and its role. Fig. 1 illustrates how typed links are used to specify the surroundings of an information unit. More precisely, the type of a UI *a* say is a second IU *b* which encapsulates code capable of handling the data structures of *a*. In this sense, the IUHM *type* is similar to the type notion of programming languages; we will return to this point in section 4.2. The role of an IU *a* is a third IU *c* which encapsulates {something} which deals with a semantics assigned to *a*. An IU may have several roles, thus enabling organizations based upon the semantic level to be set.

Beyond these two mandatory link types, several other links are useful in IUHM, and can be set by the system designer. In the implementation of OPALES[2], for example, considerable use was made of the *owner* link, but this like is not meaningful for all applications. The *inherits* link type has a strong semantics for representing Class like structures. The *relative to* link is a general-purpose link which helps define relationships between IUs. For example, a piece of code, a service *d*, might have a *relative to* link to an IU *e* whose *type* is *affinities*. In this instance, the data describing the affinities of the service *d* would be in the UI *e*, and that the code for handling these descriptions would be in the IU *affinities*. The hypertext structure is dynamic; changing a link (with respect to certain given constraints) may change the surroundings of an IU, and thereby cause other items to enter or leave the affinity of other services.

[2] As an aside, we point out that in the OPALES system, as presented in [21], the hypertext implementation was built on UI descriptors. These descriptors separated data content and links, and gave a special statute to four link types, thereby enabling faster structural computing of affinities. However, this implementation using additional links was based on practical efficiency and is specific to OPALES, rather than an aspect of the IUHM model.

A complete presentation of all the possible useful link types is beyond the scope of this paper. We point out, however, that the link mechanism may be used to derive notions found in other programming paradigms, particularly object-oriented paradigms. Notions such as simple or multiple inheritance, delegation, and so forth, may be thought of as sets of relationships, and these relationships may in turn be represented in IUHM by the use of typed links between appropriate IUs. Indeed, since the relationships in question are explicit, it becomes possible in the IUHM model to mix various techniques as required, in contrast to the situation normally found in traditional programming paradigms in which the use of a single technique is frequently enforced. Thus, one may use the type-role mechanism to depict specific inheritance rules and to selectively provide the code to compute inheritance in a given surroundings as needed.

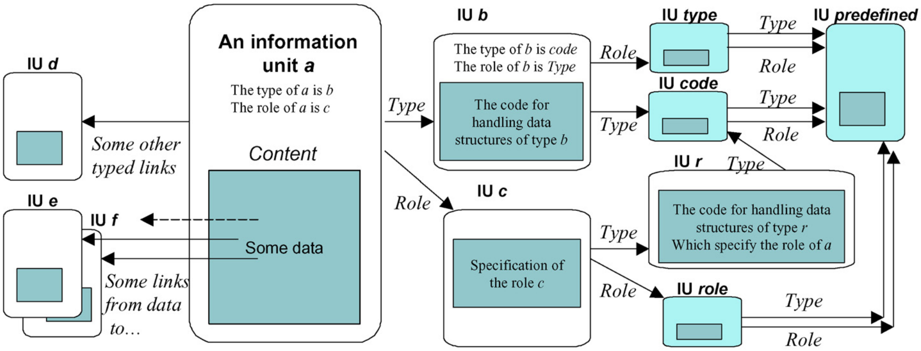


Fig. 1. Using type and role links to specify information units.

4 Architecting a Service-Based System with IUHM

In this section, we discuss how a dynamic service-based system may be modelled and implemented using the IUHM approach. It is important to realise that the complete specification of the behaviour of a system is not simply the content of IU nodes (code and data) but also includes the hypertext network induced by the links which depict relationships between IUs. As a consequence, structural reorganization of the system can in principle be achieved at low cost by editing these links[3]. In practice, this approach requires there to be in place a mechanism, the *IUHM Functional Core*, to

[3] By way of illustration, in OPALES, surroundings were used to model the concept of *workgroup*. Specifically, in OPALES surroundings are used to represent the fact that certain data is within the concern of some workgroup, and has been validated by the group moderator. Surroundings are further used to set the affinities of the services associated with the workgroup in order to specify which data the group is willing to operate on. Linking a data to the concern of a group, or moving the owner link to, say, the group moderator, changes the data surroundings, and thereby associates the specific tools dedicated to this user group with the data in question.

dynamically handle structural computing on the IU hypertext network. The IUHM functional core provides the primary mechanisms required in order to run an application described by an IUHM network[4]. To be as simple as possible, the functional core takes advantage of the reflexivity of the IUHM description. This section discusses in greater detail the notions of type, role, surroundings and affinities of IUs and describes the dynamic management of these items in the IUHM core.

4.1 Reflexivity in Type and Role Descriptions

As discussed in section 3.2 any IU *a* has a type, which is an IU *b* containing the code necessary for handling the content part of the IU *a*, and an analogous remark applies to roles. Reflexivity implies all IUs throughout the system have links to a type IU and a role IU. This type-role network is terminated by a set of *primitive* types and roles, which are directly implemented in the system core. Primitive types and roles are nonetheless represented in the hypertext network, making use of the predefined node called *predefined* (see Fig. 1). In a similar fashion all the primitive notions are represented by predefined nodes, and this includes the nodes *empty* and *undefined*, whose type and roles are themselves *predefined*. This approach ensures that the graph description is consistent with respect to link types: there is no dangling links, rather links pointing to the *undefined* node and there are no missing links, rather links to the *empty* node. This reflexive technique is both quite simple and powerful, and enables replacement and substitution of system components to be implemented by link replacement.

4.2 Why Types, Roles and Affinities?

The distinction between types and roles places emphasis on two distinct and separate aspects of the manipulation of items within the system. The type manages the technical, implementation aspects; the role determines what user-level semantics are attached to the item.

By way of an example from the digital library domain, let us consider a XML file *a*. Technically the document is simply an XML file which would have, in a classical system, the mime type *a.xml*. In IUHM, the IU *a* would have, naturally, a type link to the XML parser which is to be used in the system. The IU *a* also has a role link to a UI *b*, which might indicate, for example, that the IU *a* is an *annotation* of a segment of a movie *c* whose type is MPEG2. The role of this annotation may in turn express the *viewpoint* of an *interest group*, *d*, of ethnologists, and this group may have bound to its description an IU whose role is to set the *work rules* for its members. Further,

[4] In terms of the Dexter Hypertext Reference Model [8], most of the functional core is embedded in the *run-time* layer of the hypertext engine, and the *storage* layer consists mainly of a IU server. The *within-component* layer consists for the most part of services within system components, although a service may be far more complex than, say, a simple component presenter.

the IU which contains information about Mrs Smith, say the *group moderator*, may point to the *owner* of this set of work rules, and so on (see Fig. 2). Thus one sees that the rich semantics described by the surroundings of an IU by means of a network of role links goes far beyond the traditional notion of type.

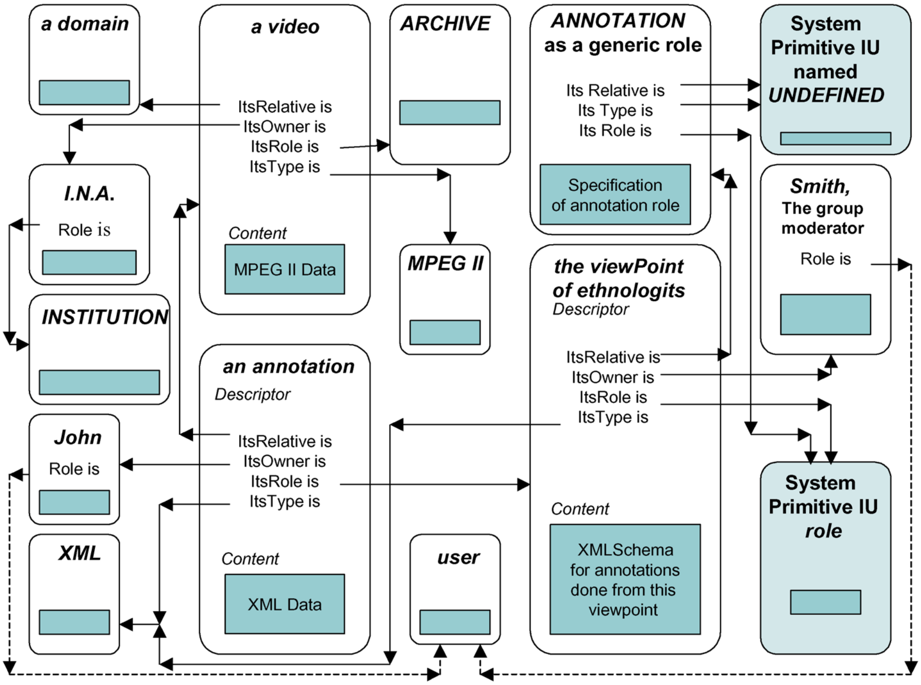


Fig. 2. Surroundings of an IU represented as a hypertext with typed links.

Furthermore, the services which can operate on a given data in a large scale dynamic system are not selected simply on the basis of the data type, which provides information at too low a level, but rather according to the surroundings of the object, which provides the appropriate semantic level. Each service can determine what part of a surroundings is significant, that is, its affinity.

Thus, continuing the previous example, the code which handles annotations is used in various services which have affinities for this code, for instance in a compound service which displays the video segment which is annotated. The annotation service provides the user with a general interface which operates on any annotation (we say that is *belongs* to the annotation role) regardless of the actual type of the annotation. Thus, both an unstructured plain-text annotation and an annotation in the form of a conceptual graph can be handled by this code, because the low level data are processed by its type IU (text or conceptual graphs in this instance, but potentially any annotation type, including XML, conceptual vectors, text, audio etc) whilst the higher level is processed by the code associated with the role.

4.3 IUHM and Reuse

The separation of type and role facilitates reuse of system components, since aspects represented by low level types and by higher level roles are clearly separated. Observe that a role is itself an IU and has therefore a type, which contains the code to handle data structures denoting the role semantics. Naturally, both type and role can be hierarchically organized with a *inherit* link. It should be noted that low level system code supporting interoperability of types and roles must be provided, IUHM does not provide any syntactic means for interoperability checking, whereas it can easily support dynamic (run time) checking.

It may be observed that data of different types may share the same role(s) without any need to adapt the role implementation. When several types share the same programming interface, various roles can be built upon this interface. In IUHM there is no need to (re-) implement an interface; rather linking an IU to a compatible role plugs this role into the underlying type. Conversely, new roles can be added by taking advantage of the type-level code, provided two conditions are satisfied:

- a role is constructed using the interface provided by the type,
- the contents of the interface are unchanged when the new type is introduced.

Fig. 3 illustrates the abstraction of a family of IUs with compatible types and roles.

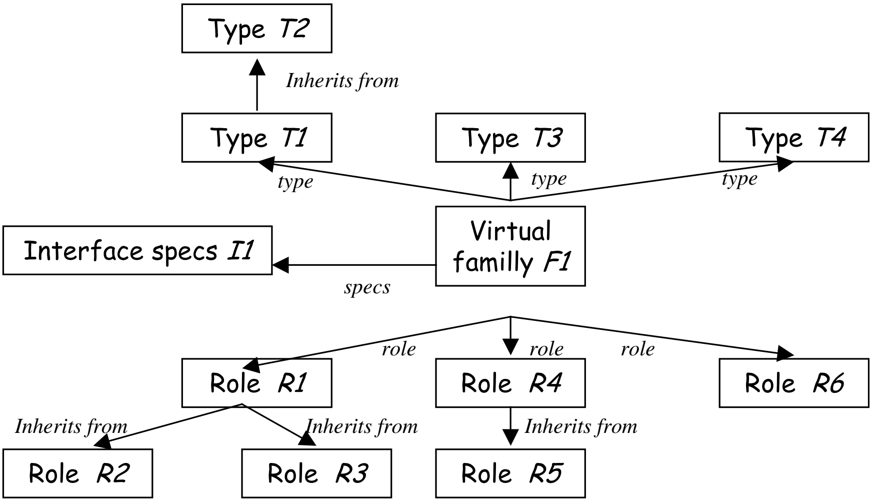


Fig. 3. Reuse of types, roles and interfaces.

The separation of type and role means that the role may be arbitrarily extended by adding new user-level operations which are either defined in terms of the defined type-role interface or which already exist at the role level. Similarly, new compatible types may take advantage of role aspects without reprogramming.

4.4 Affinities and Dynamic Aspects

IUMH manages sets of services, that is applications which cooperate and which interact with users in a given context. As mentioned in section 1, we make the assumption that interoperability between low and medium scale services is provided by classical techniques; IUHM deals only with the large-scale cooperation of services. IUHM helps in specifying and exploiting the rules of cooperation between services. Such rules are defined in terms of affinities.

A service is an IU *s* which has a link to an IU *af* whose role is *affinity* . This means that the IU *af* contains specifications which describe the affinities of *s*. We have not mentioned the type of *af* simply because the type of *af* contains the code which is capable of handling the low level description of the affinities of *s,* whereas the role *affinity* enforces the high-level methods provided by this role to comply with the *affinity* semantics predefined in the system. The *AffinityMatch (x)* method delivers a Boolean which indicates whether or not a given IU *x* has surroundings compatible with the affinity rules of the service. Affinities are usually described in terms of graph pattern matching on the hypertext network.

The IUHM functional core registers services and handles the set of affinities of the registered services in order to dispatch IUs to the appropriate services. Affinities may be quite simple, such as unitary direct links such as *type = xml*, or complex, such as *videos annotated by some of the concern of a group moderated by Ms. Smith.* Since affinity handling is managed by the general core mechanisms, there is no predefined syntax or techniques to denote affinities. The predefined core-implemented type for affinities deals only with direct combinations of types and roles, whereas one can bind more specific affinities handlers to a service just by placing a link between the affinities *af* and the type *taf* which computes these specific affinities. The essential point is that the affinities of a service must be able to answer the question as to whether an affinity is or in not interested in managing a given IU.

Another important method provided by an affinity role is *Share*. If a service responds *true* to *Share*, other services whose affinity matches the IU can share the data. For instance, a data inspector service shares the data it inspects with the other services, whilst an editor, on the other hand, may request exclusive access, to avoid contention during editing operations.

In order to deal with conflicts of interest between services, an order is defined on services at loading time. The loading of services is handled by a primary role *service loader*. By default the associated type is predefined (built-in) and would load as service any UI bound to a service role. Alternatively, one may override the default and define a private strategy for loading services by providing a link to a specific code for this type. In this fashion, the IUHM core is extremely simple; when bootstrapping, it looks for *service-loader* IUs in the description and loads these IUs, otherwise it looks for and loads services. Since, by construction, such IUs match the built in service-loader affinity, they are dispatched to the resident service loader core which installs and registers them, collecting their affinities. Initiating these non-resident service loaders induces the loading of other service, in the order and with the strategy specified by these loaders, ordering them as dictated by their affinity rules.

In summary, the notion of affinity provides a very simple reflexive mechanism within the IUHM core to define arbitrarily and to change dynamically the strategies which are used in the system. Since affinities are computed at run time, changes in the hypertext link structure may induce major changes in the system behaviour.

4.5 Affinities and Generic Service Structure

Because of the reflexive nature of IUHM, there is no difference between data and service, both are IUs. As a consequence, one service may be regarded as the data of another service. In this way service affinities can be used to denote subsets of related services (see Fig. 4).

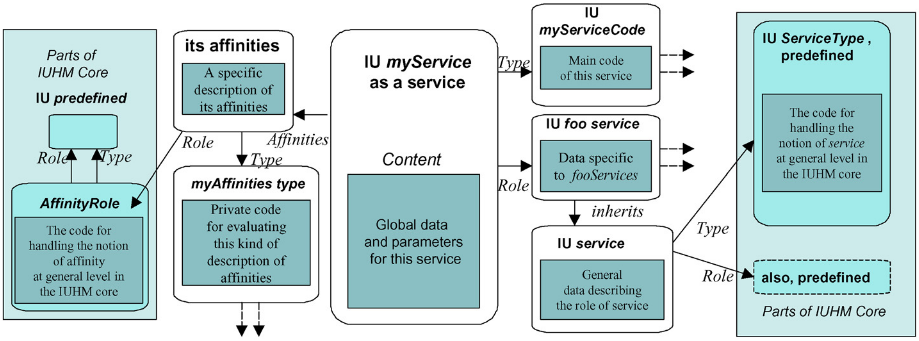

Parts of IUHM Core
IU predefined
Role
Type
AffinityRole
The code for handling the notion of affinity at general level in the IUHM core
its affinities
A specific description of its affinities
Role
Affinities
Type
myAffinities type
Private code for evaluating this kind of description of affinities
IU myService as a service
Content
Global data and parameters for this service
Type
Role
IU mySeviceCode
Main code of this service
IU foo service
Data specific to fooServices
inherits
IU service
General data describing the role of service
Type
Role
IU ServiceType, predefined
The code for handling the notion of service at general level in the IUHM core
also, predefined
Parts of IUHM Core

Fig. 4. Surroundings of a service which belongs to the *fooservice* category and which has specific affinities.

This is typically the case of the service loader behaviour, as shown in the previous paragraph, but more interesting features rely on this property. One may easily compose generic services by combining virtual services which are defined solely in terms of affinity. A compound service asks the IUHM core to call a service operating on given data, by passing as parameter the actual data and simply an affinity specifying the kind of service required. In this way the core looks for the services which match the requested affinity and from among these services, selects that which has affinities with the data to de handled. The example described in section 2.3 works in this manner. To add the *Conceptual vectors search engine* into Opales, one just needs to place a role link between this search engine and the role *search engine*. In this way, whenever the generic querying mechanism asks the core which services are in its affinity, it will receive this search engine also. Furthermore, when a compound query contains an IU whose type is *Conceptual vectors*, the generic querying mechanism would simply asks for a *search engine* to open this UI and would receive the *Conceptual vectors search engine* as the target of the core call. It may be observed that this mechanism has similarities with the polymorphism to be found in Object Programming.

5 Discussion and Related Work

In this section we discuss a number of original aspects of our IUHM Structural Computing Model for building dynamic systems. We do so within the context of the MIS Conference, and therefore we focuses our discussion on issues related to meta informatics, and we discuss how our ideas can contribute new concepts for meta development. We organize our discussion around four main points:

- object-oriented development of applications versus structural computing modelling of applications;
- openness and service-based architecture;
- open hypermedia architecture and structural computing;
- scalability, interoperability, reflexivity, flexibility, and adaptation.

5.1 Object-Oriented Development versus Structural Computing

The main idea introduced in this paper is that it is possible to represent an application architecture explicitly by a *computing* structure, compliant with the IUHM model. In this structure, any element of a system is represented by an information unit which has a type and a role. Although some similarities can be found between our approach and object-oriented programming, there are two major differences as follows:

- in object-oriented approaches, an object has a class, and methods belong to the class even when the effects of polymorphism and inheritance mean that the method is to be found be elsewhere in the inheritance tree,
- in IUHM an IU
  - has a surroundings which, by nature, depends upon other IUs and thus may change dynamically,
  - is processed by some service which is dynamically selected by the IUHM functional core from among the services which are registered at this instant in the current context, the choice being based upon the affinities declared by this set of services.

Thus the various elements which are responsible for the assignment of an IU to a service are extremely dynamic; the service loader is responsible for the set of active services in a given context, the affinities are responsible for matching data to services, the surroundings are representative of the general over structure of the system.

The main interest of using a hypertext structure to denote these relationships is to offer a very flexible technique to separate clearly the concerns of data, algorithms, and the concerns of system structure (the HTX structure).

Our approach can be thought as an extension of the ideas of Adaptive Object Models-AOM- [2]. Adaptive Object Models provide a way to free the designer from the obligation of creating dozens of classes when it is not necessary. An AOM is basically an instance-based model in which some instances play the role of classes (similar to types in IUHM) and others play the role of base objects. AOMs use the

Type Object pattern [10] and the Strategy pattern [7] to provide a way to add behaviors dynamically.

The main difference between AOMs and the IUHM is that while the former is still based on a "traditional" separation between classes and instances, the latter introduces the idea of roles to separate clearly the basic operations of an object (specified in its type) from the semantics of that object from the user's points of view (specified in the role).

In [28] it is shown how to implement roles using a conventional object- oriented language. The Role Object pattern use decorators [7] to "extend" a base class with roles.

The difference between types (or classes) and roles has been widely discussed in the literature. In [12] the authors propose to include roles as first-class citizens in class-based languages. In this proposal, roles permit an object to behave differently when playing different roles.

The OORAM (object-oriented role analysis and modelling) software engineering method [26] proposes to use the concept of roles from the early stages of the software life cycle. While we are not focusing on analysis and modelling, many of the ideas in OORAM can be applied while building IU networks.

Finally, [28], [29] discuss how to use roles to describe and design composite patterns and object-oriented application framework. The authors introduce the concept of Role Diagrams and show that different class-based implementations can be derived from these diagrams. In this case roles are viewed as higher-level abstractions that allow simplified descriptions of complex object interactions.

5.2 Openness and Service-Based Architecture

From the considerable literature on the subject, it is clear that the construction of open systems has been a topic of great interest for some time. Various techniques have been proposed to deal with different abstraction levels, from hardware levels with techniques such as plug-and-play devices to, more recently, business levels. Furthermore a major constraint is to be able to deploy networked applications. The current trend is to design service-based architectures which enable to separate concerns of services offered through the application and concerns of components involved to offer the various services [32]. Jini network technology [9] is an open software architecture that enables developers to create network-centric services -- whether implemented in hardware or software -- that are highly adaptive to change. Jini technology can be used to build adaptive networks that are scalable, evolvable and flexible, as typically required in dynamic computing environments. Jini is oriented towards development. The growing movement around web services, the new step in the evolution of the World Wide Web infrastructure, aims at allowing programmable elements to be placed on Web sites where others can access distributed behaviors through published description of services (WSDL) [33]. However, these descriptions do not appear sufficient to elaborate strategic development for business applications. The UDDI registries [30] are used to promote and discover these distributed Web Services, by including explicit description of business models. But from a design point of view, one can observe that there has been an evolution from designing application by decomposition [5], [31], [1], to designing application by composition [3], [25], or by flow description [14]. One should observe that standards are emerging for describing services offered by distributed components et clearer interface to "plug" them into applications but there is not yet clear support for explicitly modeling both technical conditions of behaviour and semantic conditions of use. Furthermore, models of composition of web services are still as yet the object of reflexion [34]. Whereas IUHM is still in its infancy, it offers both a technical architectural and executable infrastructure for integrating open services, data and metadata and a way to explicit as a separate hypermedia network syntactic and semantic constraints (through types, roles and so forth), thereby providing explicit modelling of application structure and behaviour. Openness and dynamic mechanisms have also been discussed in [21]. In so far as there exists an explicit structure, there is possibility of structural computing for various purposes; we use the term *meta-computing* for this concept.

5.3 Open Hypermedia Architecture and Structural Computing

These computing fields are very representative of growing efforts in a particular domain, hypermedia, to use generic architecture [35] in order to separate the concerns of application modelling and underlying techniques used to manipulate the hypermedia structures of the application [27]. However, as we described in [21], we feel that an approach which adopts distinct models for describing hypermedia structures on the one hand and services on the other are not relevant. The use of such an approach in our view complicates the management of openness and interoperability while maintaining homogeneous semantics [18]. Indeed, a matter of ongoing research for us is how we can utilise IUHM to model various structural domains as a single general-purpose hyper-structure. Our approach to this question is a two-level one. At the first level, we would define a set of *model patterns,* which model various structural domains in terms of IUHM graphical structures, each pattern corresponding to a particular structural domain. This stage appears to us to be quite feasible, and indeed, were we not able to do this first-level modelling, it would indicate a shortcoming within the IUHM model. The second stage is more difficult, and would describe the various model patterns themselves in terms of a single IUHM structure. Since IUHM is reflexive, this second-level modelling is theoretically feasible, but the details require considerable further deliberation.

5.4 Scalability, Interoperability, Reflexivity, Flexibility, Adaptation

A primary requirement of architecture capable of describing a dynamically evolving system is that the architecture should embody a simple resolution of the problems of *interoperability* between modules. The term interoperability refers to commonality of access means for services in all domains, and is distinguished from, say, the provision of middleware components specifically related to particular domains, such as one finds in RPC or CORBA. A number of approaches to interoperability are to be found in the literature, including object-oriented approaches [22], 15], [16], layered approaches [17], and aspect oriented programming [11], [6]. While our approach has elements in common with several of these, we have not found any of these existing approaches entirely adequate for our needs. These various approaches appear to be more concerned with applying these notions to implementation, whereas our perspective embodied in IUHM is that an IUHM compliant structure makes it possible to both model and support execution of the application.

We have focused on the notion of interoperability, which is one of the most important quality criteria for software. We observe that IUHM is fully reflexive, thus facilitating adaptation and offering flexibility. Owing to its reflexivity, IUHM also support scalability of modelling and development.

6 Conclusion

The IUHM technology is the consequence of a three-year maturation of the Opales project, in which we had to cope with the continuous evolution and enrichment of the system, and its adaptation to the evolution of user needs. The first version of Opales which has been initially developed using classical programming techniques made us conscious of the need for a flexible architecture for user configurable service based systems. Our long experience in hypermedia systems suggested to us that we should take advantage of typed-links hypertext structure and of structural programming to support the specification, the development, and the evolution of the system in a unified manner. IUHM is the result of the experience gained in this long development. In this paper we have gone beyond the OPALES experience, and have extracted the key elements which may be utilized to depict and organize large-scale service based applications in a generic manner.

Representing relationships between the components of the application structure as a typed-links hypertext graph provides a simple and flexible approach to the description of system composition and of application architecture evolution. IUHM provides a means to handle late binding between any entities in the system, relying on surroundings and affinities, both of which reflect dynamic aspects of the system. IUHM sets a paradigm both for the description and for the dynamic behaviour of the system. We observe that the reflexive architecture of IUHM adds a great deal of flexibility in the design, which enables any of its own mechanisms to be overridden in accordance with the designer's choice. Even the service affinity determination code or the service loader code themselves are handled as services and thereby can be overridden at will, as by editing links in the IUHM description.

Many other techniques, of course, are available to design and implement large scale service-based applications. The major difference between such techniques and IUHM relies on the orderly separation of three major aspects of a system, its technical aspect (types), its functional code (roles), and the relationships between services, data and any notions in the system (IUHM graph). This separation is the key to code reuse and sharing and enables the flexible reorganization of the overall architecture by simply changing the IUHM description.

References

1. Architecture Board MDA Drafting Team. (2001). Model Driven Architecture a technical perspective. Document Number ab/2001-02-01.
2. AOM. See http://www.adaptiveobjectmodel.com.
3. Atkinson C., Bayer J., & Muthig D. Component-based product line development. *Software product lines: Experience and research directions*. Edited by Patrick Donohoe. Kluwer Press. ISBN 0-7923-7940-3, 2000.
4. Betaille, H., Nanard, J., & Nanard, M. OPALES: An Environment for Sharing Knowledge between Experts Working on Multimedia Archives. In *Proc. Conf. Museums and the Web*, Seattle, 2001, 145-154.
5. Bredemeyer, D. & Malan, R. Software architecture, central concerns, key decisions, 2002, http://www.bredemeyer.com/pdf-files/ArchitectureDefinition.PDF
6. Constantinides, C.A., Bader, A., Elrad, T.H., Fayed, M. E., & Netinand, P. Designing an Aspect-Oriented Framework in an Object Oriented Environment. *ACM Computing Surveys*. March 2000.
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*, Addison Wesley, Reading, 1995.
8. Halasz, F. & Schwartz, M. The Dexter Hypertext Reference Model , *NIST Hypertext Standardisation Workshop*, Gaithersburg, 1990, also in *CACM*, Vol. 37 (2), (version without specification in Z), 1994, 30-39.
9. JINI, see http://www.jini.org
10. Johnson, R. & Woolf, B. The Type Object Pattern. http://www.ksc.com/article3.htm
11. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., & Irwin, J. Aspect-oriented programming. In ECOOP'97, Object-Oriented Programming, 11th European Conference, LNCS 1241, pages 220--242, 1997.
12. Kristensen, B.K., Osterbye, K. Roles: Conceptual Abstraction Theory and Practical Language Issues. *TAPOS 2(3)*: 143-160, 1996.
13. Lafourcade, M. Guessing Hierarchies and Symbols for Word Meanings through Hyperonyms and Conceptual Vectors. *In Proc. of OOIS 2002 Workshop*, Montpellier, France, September 2002, Springer, LNCS 2426, 84-93.
14. Leymann F. (2001). Web Services Flow Language. IBM Software Group. *WSFL 1.0*.
15. Liu, L. & Pu, C. The distributed interoperable object model and its application to large-scale interoperable database systems. In *Proc. ACM Int'l. Conf. on Information and Knowledge Management*, ACM Press, 1995.
16. Manola, F. & Heiler, S. An Approach To Interoperable Object Models, In *Proc. of the International Workshop on Distributed Object Management*, Edmonton, Canada, August 1992.
17. Melnik, S. & Decker, S. A Layered Approach to Information Modeling and Interoperability on the Web. In *Proc. ECDL'00 Workshop on the Semantic Web*, Lisbon, Portugal, Sept 2000.
18. Millard, D.E. & Davis, H.C. Navigating spaces: the semantics of cross domain interoperability, *Proc. 2nd Int. Workshop on Structural Computing*, Springer-Verlag, LNCS 1903, 2000.
19. Nanard, J. & Nanard, M. Using types to incorporate knowledge in hypertext. In *Proc. ACM Conf. Hypertext'91*, ACM Press, 1991, 329-344.

20. Nanard, M. & Nanard, J. Cumulating and Sharing End-Users Knowledge to Improve Video Indexing in a Video Digital Library. In *Proc. ACM / IEEE Joint Conf. on Digital Libraries*, ACM Press, 2001.
21. Nanard, J., Nanard, M., & King, P. IUHM, a Hypermedia-Based Model for Integrating Open Services, Data and Metadata. In *Proc. ACM Conf. Hypertext 2003*, ACM Press, 2003.
22. Nguyen, T. Towards Document Type Evolution - An Object-Oriented Approach, *In Proc. of AusWeb'02 Conference*, 2002.
23. Nürnberg, P.J., Leggett, J.J., & Schneider, E.R. As we should have thought. In *Proc. ACM Conf. Hypertext'97*, ACM Press, 1997, 96-101.
24. Nürnberg, P.J. & Schraefel, M.C. Relationships among structural computing and other fields. *J. of Network and Computing Application*, special issue on Structural Computing, October 2002.
25. Oellermann, W.L. Architecting Web Services. A Press. ISBN 1893115585, 2001.
26. Reenskaug, T. *Working with Objects*. Prentice Hall, 1996.Uniform Description Discovery and Identification: http://www.uddi.org
27. Reich, S., Wiil, U.K., Nürnberg, P.J., Davis, H.C., Gronbæk, K., Anderson, K.M., Millard, D.E., & Haake, J.M. Addressing interoperability in open hypermedia: the design of the open hypermedia protocol. *The New Review of Hypermedia and Multimedia*, 1999, 207-248.
28. Riehle, D. Composite Design Patterns. In *Proc. OOPSLA'97*, ACM Press, 218-228, 1997.
29. Riehle, D. & Gross, T. Role Model Based Framework Design and Integration. In *Proc. OOPSLA 98*, ACM Press, 117-133, 1998.
30. Uniform Description Discovery and Identification: http://www.uddi.org
31. UML, see http://www.omg.org/technology/documents/formal/uml.htm.
32. Van Zyl, J. A perspective on service-based architecture: the evolutionary concept that assists technology providers in dealing with a changing environment. In *Proc. SAICSIT 2002*, 2002.
33. Web services: see http://www.w3.org/ws
34. Web Services Choreography working Group: see: http://www.w3.org/ws/choreography group
35. Wiil, U.K. Toward a proposal for a standard component-based open hypermedia system storage interface. In *Proc. OHS6 and SC2*, LNCS 1903, Springer Verlag, 2000.

Structuring Cooperative Spaces: From Static Templates to Self-Organization

Jessica Rubart[1] and Thorsten Hampel[2]

[1] Fraunhofer Institute for Integrated Publication and Information Systems (IPSI)
Dolivostrasse 15, 64293 Darmstadt, Germany
rubart@ipsi.fhg.de

[2] University of Paderborn, Department of Computer Science
Fürstenallee 11, 33102 Paderborn, Germany
hampel@uni-paderborn.de

Abstract. Cooperative spaces can be viewed from the three structural dimensions *process*, *team*, and *content*. Each of these dimensions can range from static templates to self-organization. As a result, we get a groupware classification based on structural abstractions. This classification supports groupware development. Additionally, one can identify more easily similarities between groupware and other fields with respect to the three structural dimensions.

1 Introduction

Groupware aims at improving the cooperation between humans in terms of efficiency and effectiveness. Ellis et al. [2] define groupware as "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment." Usually, groupware users are provided with a kind of cooperative space. Groupware research is distributed across different domains, such as Computer Supported Cooperative Work (CSCW) or Computer Supported Collaborative Learning (CSCL).

From the *metainformatical view* [9] taken in this community people use groupware to support their organization's primary task. Thus, groupware belongs to the B level of Engelbart's ABC model of organizational improvement [3]. From this perspective, developing groupware is B level work. This B level work in turn can be improved by providing for example development frameworks or methodologies as C level work.

In this paper, we contribute a structure-oriented way for classifying cooperative spaces, which we call the *Graz Groupware Structure (Graz GS) space*. This classification can be considered C level work as it supports developers in taking into account the different structural dimensions when defining the flexibility of cooperative systems. In addition, one can identify more easily similarities to other fields with respect to the three structural dimensions, i.e. we might be able to classify concepts or systems of other fields according to (some of) these dimensions as well.

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 119–125, 2004.

2 A Structure-Oriented Groupware Classification

In cooperative spaces there are three main structural abstractions: The process structure, the team structure, and the content structure [5].

The process structure considers processes, which can be partitioned into several activities that in turn can contain a number of work items. In addition, it addresses the dependencies between the different structural abstractions. The team structure represents the cooperating users, the teams, the different roles, which users and teams can take, and their relationships. The content structure, finally, models the contents of the cooperative space, such as shared documents and their relationships. In addition, the process, team, and content structures are interconnected in order to express, for example, that team X is assigned to work on task Y.

Thus, the process, team, and content structural abstractions are different enough from each other to form important dimensions of cooperative spaces.

In the following, we show that each of these three dimensions can be *well-structured*, *semi-structured*, or *self-organized* and thus ranges from static templates to self-organization. As a result, a groupware structure space can be spanned, which is shown in Fig. 1, and cooperative spaces can be classified accordingly.

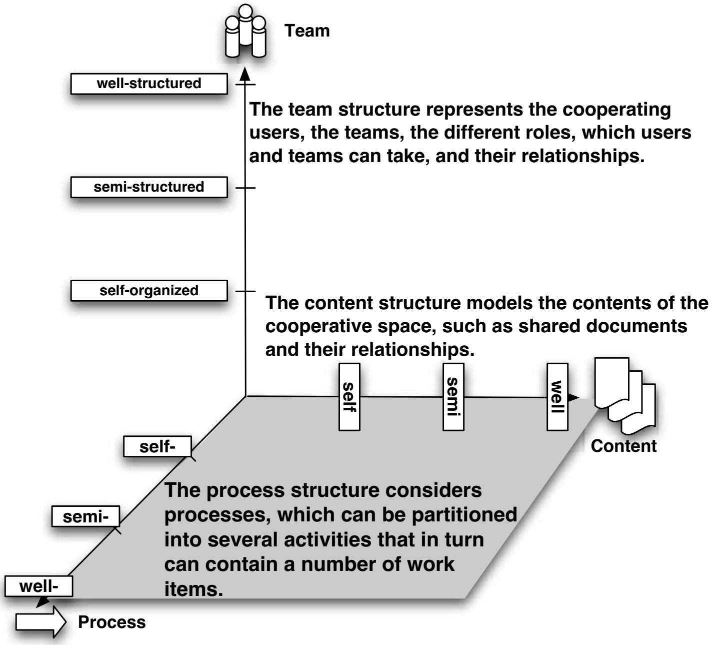

Team
well-structured
semi-structured
self-organized
The team structure represents the cooperating users, the teams, the different roles, which users and teams can take, and their relationships.
The content structure models the contents of the cooperative space, such as shared documents and their relationships.
self
semi
well
Content
self-
semi-
well-
The process structure considers processes, which can be partitioned into several activities that in turn can contain a number of work items.
Process

Fig. 1. Groupware Structure Space

2.1 Process Structure

The process structure is the most well-known structural dimension. It is explicitly addressed in the context of workflow management systems. Marshak [8] states: "a well-established workflow enables routine processing while an ad hoc workflow enables exception handling". Production workflows are usually well-structured and don't need much change. Semi-structured workflows are partly pre-structured, but also have a flexible part. Finally, ad hoc workflows are initially almost un-structured and evolve during runtime and thus are basically self-organized. In addition, people need to be able to interact with each other and the system in order to organize a running workflow [11]. Creativity tools like one for brainstorming or one for cooperative mind mapping support ad hoc activities.

In the collaborative learning domain there are so-called learning protocols [10], which specify the flow of synchronous learning units and require all learners being online. Thus, the process domain of such protocols is well-structured. The following collaborative learning scenario at universities, for example, can be characterized by a semi-structured process: In the context of a course students work on weekly assignments and have to meet a deadline when passing the results to the tutor(s). The tutor(s) then correct the results. So there is a pre-structured process, but within this process there is a lot of flexibility, such as how to work on the assignments.

2.2 Team Structure

In the above-mentioned university example the team structure is either semi-structured or self-organized. Courses themselves are usually predefined. In addition, a course might be divided into subgroups, which means that it is pre-structured. Students register themselves for a course, which can be considered self-organization. If the system then automatically creates subgroups, this is again pre-structuring. However, if the students can register for participating in a group and can be responsible for administering group areas, then this is again self-organization [6]. There are also CSCL environments based on well-structured teams [13]. Here, either the tutors create groups manually, or the system automatically creates groups based on the learners' contexts, e.g. their paths through the course.

In workflow management systems roles and persons can be explicitly assigned to tasks, which implies that teams are well-structured. In other cases - even if there are well-structured processes - the team structure might not be well-defined. It might be that there are just roles assigned to different activities and that at run-time persons have to be found and teams have to be established for working on the activities [11]. This again defines a range from well-structured to self-organized.

2.3 Content Structure

The historical development of computer supported applications for cooperative handling of work material has been predominantly shaped by CSCW systems designed mainly for office applications [4]. The focus here is largely on document management functions that allow documents to be organized and processed according to strictly defined rules and user rights arrangements. The most extreme example of this can be found in workflow management systems, in which the path documents take between different cooperation partners can be prescribed.

However, even if the processes are well-defined, the document flow doesn't necessarily have to be. With reference to creative activities [11], for example, people might not know in advance, which tools they will use, which documents will be created, and which documents need to flow to other activities.

Shared workspace systems, such as BSCW[1] [1], support a semi-structured content structure. Workspaces can be created and people who should have access can be invited. Workspaces can be organized hierarchically. Access rights can be used explicitly to restrict the access to documents, folders, and workspaces.

The system sTeam[2] [6] focuses on self-organization and self-administration of virtual knowledge areas (see Fig. 2). Based on the metaphor of rooms users can create and occupy virtual knowledge areas and move between them by means of gates. Virtual knowledge areas support the organization of material and the cooperation between users. Access tools provide different views of areas and support different kinds of organization. A shared whiteboard, for example, supports synchronous cooperative structuring and annotation of knowledge areas.

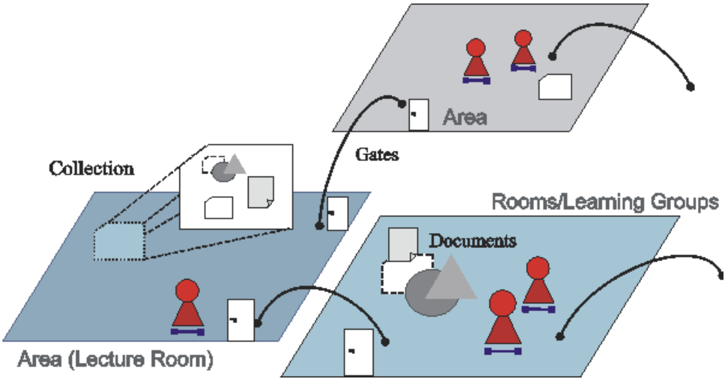


Fig. 2. Virtual Knowledge Areas in sTeam

In addition, users of sTeam don't deal with access rights explicitly. Rather, these are dependent on the cooperation context. An example of this is the passing on of a document from one user to another. Here, the rights are dynamically adapted to the document's environment. For instance, if the first user has read and write access rights, these rights (e.g. a write access right) change in favour of the second user if the document is passed on.

[1] "Basic Support for Cooperative Work"; http://bscw.gmd.de/

[2] "structuring information in Teams"; http://www.open-steam.org/

2.4 Benefits of the *Graz GS Space*

The proposed classification schema accommodates users and developers of groupware with awareness about the different structural dimensions when using and developing groupware, e.g. by assembling software components or building new ones to create a groupware solution. Dependent on the use case, which is supported by a groupware solution, different kinds of flexibility related to the structural dimensions are required. These should be made explicit when designing a groupware solution as part of the requirements analysis in a software engineering process.

A groupware infrastructure that addresses lots of different usage scenarios should support the whole structure space as visualized by Fig. 3. This means that such an infrastructure needs to provide configuration possibilities or variation points for developers so that it can be tailored and potentially extended to support a specific usage scenario that might be limited to (a) specific part(s) of the space.

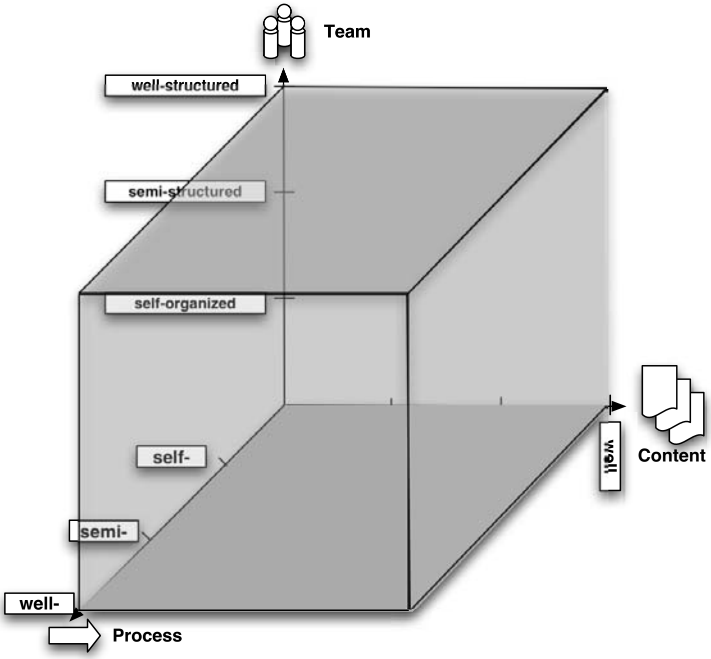


Fig. 3. Structure Space of a Groupware Infrastructure

Moreover, the *Graz GS space* provides different criteria for comparing groupware solutions than other classification schemas do. This enables the identification of distinctions between systems, which otherwise seem similar. For example, both sTeam and BSCW support the organization of documents by providing specific shared workspaces. However, as discussed in section 2.3 BSCW focuses on a semi-structured content structure while sTeam on self-organization.

For sTeam to provide self-organization related to the content and team structures we found its concept of “contextual access rights” important. If a participant in a discussion, for example, is appointed to the role of the moderator, she automatically acquires the access rights to documents and materials, or to the discussion environment, that go with this appointment.

The *Graz GS space* doesn't rate any specific groupware solution as different solutions address diverse use cases. Nor does it provide hard criteria when to call a solution's structural dimensions well-structured, semi-structured, or self-organized. Rather, it makes explicit that these structural differences exist and can be used to compare systems and concepts.

Finally, this classification schema supports the identification of similarities to other fields according to the three structural dimensions. For instance, search engines can be compared to document management systems with respect to the structuring flexibility of the content dimension, i.e. in what way the content structure (like the presentation of search results or documents) is pre-structured and can be self-organized by the users.

3 Conclusions

We have shown that all of the three structural dimensions *process*, *team*, and *content* of cooperative spaces can be characterized as *well-structured*, *semi-structured*, and *self-organized*. Even if processes are well-defined, in a workflow management system for example, the team and content structures might be semi-structured or self-organized. This results in a groupware structure space, which can be used for classifying systems and concepts. Furthermore, it provides support for developing groupware solutions as considering the different kinds of flexibility related to the structural dimensions is an important part of requirements analysis. In addition, the *Graz GS space* supports the comparison of systems and concepts along the structural dimensions.

There are existing classification schemas for groupware, such as the classical time and space matrix [7] or the application level classification [2] that has categories like message systems, group editors, electronic meeting rooms and so forth, or the “C-oriented” one, which classifies systems depending on the intensity of their support for communication, coordination, and cooperation [12]. The classification schema presented in this paper is complementary to the existing ones. In addition, existing classification schemas for groupware do not focus on structural abstractions and are thus not that useful for identifying similarities to other fields as the proposed classification schema is. Therefore, the proposed classification schema supports the metainformatical mood.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments and the attendees of the Metainformatics Symposium 2003 in Graz for valuable discussions. In addition, we want to thank Thomas Bopp for improving our figures.

References

1. Bentley, R., Horstmann, T., Sikkel, K., and Trevor J. 1995. Supporting Collaborative Information Sharing with the World Wide Web: The BSCW Shared Workspace System. *The World Wide Web Journal: Proceedings of the WWW Conference*, Issue 1, O'Reilly, 63-74.
2. Ellis, C. A., Gibbs, S. J., and Rein, G. L. 1991. Groupware - Some Issues and Experiences. *Communications of the ACM*, 34 (1), 38-58.
3. Engelbart, D. C. 1992. Toward High-Performance Organizations: A Strategic Role for Groupware. *Proceedings of the GroupWare '92 Conference*, Morgan Kaufmann Publishers.
4. Greif, I. 1988. Computer Supported Cooperative Work: A Book of Readings. San Mateo: Morgan Kaufmann Publishers. 5.
5. Haake, J. M. 1999. Facilitating Orientation in Shared Hypermedia Workspaces. *Proceedings of Group'99*, ACM Press.
6. Hampel, T., and Keil-Slawik, R. 2003. Experience With Teaching and Learning in Cooperative Knowledge Areas. *Alternate Track Proceedings of WWW 2003*, MTA SZTAKI, 125-132.
7. Johansen, R. 1988. Groupware: Computer Support for Business Teams. The Free Press, 12-45.
8. Marshak, R. T. 1995. Rethinking Workflow - Part II: Routine Processing vs. Exception Handling. *Workgroup Computing Report*, 18(4), 2.
9. Nürnberg, P. J. 2003. Building Metainformatical Bridges. *Metainformatics. International Symposium*, MIS 2002, Springer-Verlag, 6-8.
10. Pfister, H.-R., and Mühlpfordt, M. 2002. Supporting discourse in a synchronous learning environment: The learning protocol approach. *Proceedings of CSCL 2002*, Hillsdale, Erlbaum, 581-589.
11. Rubart, J., Wang, W., and Haake, J. M. 2003. Supporting Cooperative Activities with Shared Hypermedia Workspaces on the WWW. *Alternate Track Proceedings of WWW 2003*, MTA SZTAKI, 243-251.
12. Sauter, C., Mühlherr, T., and Teufel, S. 1994. Sozio-kulturelle Auswirkungen von Groupware - Ein Ansatz zur Adaption und Operationalisierung eines sozialpsychologischen Modells für die Gestaltung und den Einsatz von Groupware. *Tagungsband des vierten Internationalen Symposiums für Informationswissenschaft*, Universitätsverlag Konstanz, 517-526 (in German).
13. Wessner, M., and Pfister, H.-R. 2001. Group Formation in Computer-Supported Collaborative Learning. *Proceedings of Group'01*, ACM Press, 24-31.

Dynamic Personalization in Knowledge-Based Systems from a Structural Viewpoint

Armin Ulbrich, Dolly Kandpal, and Klaus Tochtermann

Know-Center Graz, Inffeldgasse 21a
8010 Graz, Austria
{aulbrich,dkandpal,ktochter}@know-center.at
http://www.know-center.at

Abstract. Knowledge-based systems support knowledge workers in those areas of their daily work which are crucial to achieve the business goals of an organization. An open scientific challenge is the derivation of organizational knowledge from the individual knowledge of the knowledge workers, and vice versa, to provide knowledge workers with the organizational knowledge they need for performing best their daily tasks. One approach to address this challenge is to design knowledge-based systems in a way which allows them to personalize the organizational knowledge to a variety of employees according to their personal needs. In this context, often "static" personalization concepts were developed in the past. Static personalization, however, does not take into account that an individual's knowledge changes over time. The idea of this paper is to extend the traditional approach of static personalization towards "dynamic" personalization in the context of knowledge-based systems.

1 Introduction

Knowledge-based systems and knowledge management systems have been around for a couple of years and tremendous research efforts have been undertaken in order to make the systems' functionalities as supportive for knowledge workers as possible. In our notion, Knowledge-based systems are generic computer systems which manipulate and provide digitized artifacts representing knowledge. Knowledge Management systems can be seen as specific instantiations of knowledge-based systems.

Knowledge management is a broad field of research which touches many different scientific areas, including computer science, economy, psychology and human resource management. Due to this the development of knowledge management has been coined by many different research areas – and this is why knowledge management is so interesting for the field of meta-informatics.

The aim of this paper is not only to present our research results in the field of dynamic personalization for knowledge-based systems. Moreover we want to put our research results in the context of structural computing. Structural Computing aims at deconstructing the relationship between data and structure: Traditionally, in computing data is considered more important than structure. Structural Computing aims at giving structure first-class status and is concerned with finding technical means to make software systems essentially structure-enabled. This paper inspects if

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 126–142, 2004.

our current research results are still plausible, correct and coherent when considered from a structural computing perspective, i.e. the focus is more on structure than on data.

The paper is structured as follows: In chapter 2 we will briefly describe research in personalization in Knowledge Management and in Structural Computing. In chapter 3 we will present the meta-model that has been designed for enabling knowledge-based systems to dynamically personalize knowledge. In chapter 4 we will present the system architecture that allows for dynamic personalization in knowledge-based systems. In chapter 5 we closely examine the given meta-model from the viewpoint of Structural Computing. Chapter 6 concludes with a brief summary of our findings.

2 Related Work

Our work is related to a wide area of other scientific work. There are aspects of Knowledge Management that are strongly related to our work, there are issues coming from research in personalization and there are certain aspects within our work that appear to be structurally coherent to specific aspect in Structural Computing. This section briefly puts our research in the context of existing research.

2.1 Personalization in Knowledge Management

Generally speaking, knowledge work contains aspects from Engelbart's three levels of work. These three levels are A-level work (directly related to an organization's primary function), B-level work (providing tools and support for A-level work) and C-level work (supporting B-level work): Knowledge workers need to improve their work practices and adapt them to emerging needs and requirements. A promising way of improving the performance of work practices is to track and enhance the basis on which the respective level of work relies. If seen from top-down, A-level workers rely on the tools and methods that are provided on the B-level of their work, and the B-level accordingly relies on its foundation, the tools and methods of C-level work. For tools and methods to be made visible and available, knowledge workers need (among others) to be supported by means of Knowledge Management, which – according to [09] –accomplishes the following tasks:

1. Provide a knowledge-friendly environment in which knowledge can develop and flourish
2. Provide knowledge workers with context-sensitive knowledge
3. Support knowledge workers in applying their knowledge for action

However, there are three reasons why today's Knowledge Management systems almost never fulfill all three tasks:

Firstly, Knowledge Management heavily relies on the underlying organizational memory, i.e. the entirety of the knowledge which exists in an organization. The organizational memory consists of knowledge that has been made explicit and formalized. In addition, it also consists of knowledge in the heads of the employees. The disadvantage of organizational memories is that they generalize individual knowledge by ‘digitizing’ human knowledge and therefore are in peril to lose several important aspects of the internal models behind an individuals’ knowledge.

Secondly, knowledge workers see themselves often in situations in which they are faced with the so-called knowledge gap. The knowledge gap is mismatch between the knowledge a Knowledge Management Systems offers a knowledge worker and what the knowledge worker requires and expects. There are three reasons for this phenomenon: The knowledge supplied lacks context, the knowledge is out-dated or irrelevant or too generic meta-data is assigned to it. The diversity concerning the knowledge worker’s needs is not met by the single, general model of knowledge supplied by the Knowledge Management system.

Thirdly, we are currently facing a trend towards mobile computing. Knowledge workers are accessing the organizational memory using a variety of diverse devices, each of which needs the knowledge to be presented in a specific representation style.

To overcome these drawbacks, the concept of personalization can be applied [9]. Still, the personalization concept of [9] covers only static situations. That is, it is not flexible enough for taking into account that a knowledge worker continuously builds up new knowledge, skills, competences etc. Also, it is not strong enough to be applied for displaying knowledge from an organizational memory on a broad range of devices (e.g. mobile phone, PDA, portable, PC etc.). Therefore we extended our concepts for static personalization towards dynamic personalization. In chapter 3 we will introduce a meta-model which supports dynamic personalization in knowledge-based systems.

2.2 Structural Computing

The term structural computing is used to denote a specific notion of computer science where the traditional relationship between data and structure is deconstructed. Traditionally, data is considered to be the main ‘ingredient’ of information systems: It contains all relevant information and transports the meaning of the stored information. Structure on the other hand only serves to connect pieces of data together, for instance to allow for traversal between several data items.

Structural Computing more focuses on the infrastructure behind collections of data and puts the research focus on relations between objects rather than on the independent, discrete objects themselves ([6]). Structure and data are considered to be equally important and structure needs to be emancipated from data’s traditional predominance. Hence, the basic underlying principles and paradigms are quite different to the traditional data-centered approach.

When looking at a software system from a structural viewpoint, one essential question is: What is the structural abstraction in that system? What are the atoms or primitives that make up the most basic structural entities in the system? What do the higher-level structural abstractions look like? Another essential question is concerned with the structural transformation. If a structure changes (over time or due to external or internal events), it needs to be transformed into a different structure. The system provides operations in form of behaviors that allow for transforming structures into one another. The question to ask is what are the most essential structural transformations in a given domain or context? The third essential question is “where is the structure stored, processed or presented”; on the client side, in the middleware, at the backend side of the system or even on operating system level?

In chapter 5 we will discuss the meta-model from chapters 3 and 4 in the light of Structural Computing. The objective is to check whether the concepts for dynamic personalization in Knowledge Management meet the requirements from a different domain with its different mind-sets and paradigms. We will examine how the structural abstractions and transformations of model for personalization in Knowledge Management look like and how to deal with several issues that arise from that investigation. Still, we will not address questions concerning where to store, process and present structure in this article, but it surely needs to be considered in future research.

3 A Meta-model for Dynamic Personalization

One possible way of achieving dynamic personalization of knowledge-based systems is to design a generic meta-model for dynamic personalization that can be used to derive domain-specific models for various domains like Knowledge Management, eLearning, Hypermedia etc. This section briefly presents the meta-model for dynamic personalization proposed by one of the authors as part of her doctoral thesis work. We use the definition of Tekinerdoğan [7] who defined a meta-model as abstractions of a set of existing models. The proposed meta-model consists of three basic entities: (1) the UserContext, (2) the PersonalizationEngine, and (3) the PersonalizableEntity. The diagram in Fig. 1 shows the interrelationship between the three basic entities:

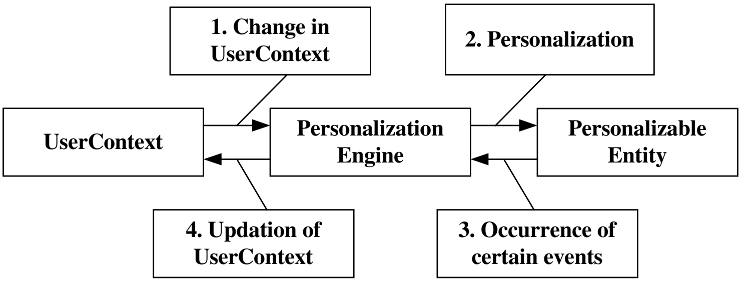


Fig. 1. The basic entities of the proposed meta-model

Changes in UserContext lead to Step 1 (Change in UserContext) that triggers the PersonalizationEngine. This in turn leads to Step 2 (Personalization) that leads to the personalization of the PersonalizableEntity. Step 3 (Occurrence of certain events) occurs as a result of triggering of certain Event Condition Action (ECA) Rules which are stored in the meta-model and deal with occurrence of certain events related to User navigation and other user triggered activities, like change in user role, achievement of certain milestones, and so on. This step leads to the triggering of the PersonalizationEngine, and by means of Step 4 (Updatingf UserContext) leads to updating the UserContext, leading to dynamic event-based personalization by means of Steps 1 and 2 again.

The following brief description of the three basic entities and the meta-model will help in elaborating the concept described above:

UserContext: Each user is associated with a UserContext that encompasses not only the user profile related information (such as their interests, role, knowledge, interaction history, skills), but also the environment related information (such as time, location, device, network, and application software related characteristics). All the entities like time, location, device characteristics, identity, role, skills etc. are classes with specific attributes and properties of their own, inheriting the generic attributes from their respective parent classes, UserProfile and Environment, as shown in Fig. 2:

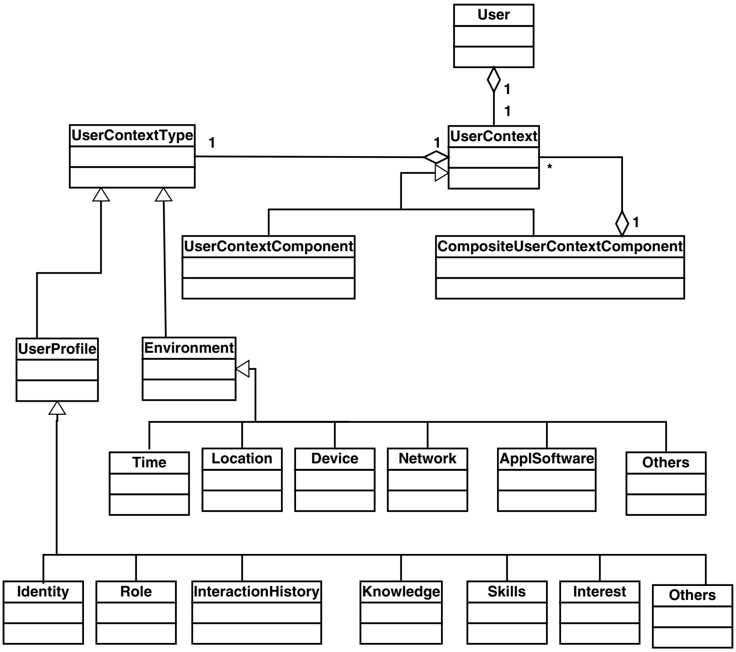


Fig. 2. User Context

PersonalizableEntity: A PersonalizableEntity is any entity that is the target of dynamic personalization. Changes in the UserContext or other user-induced changes lead to personalization of such an entity. The following are the subtypes of the PersonalizableEntity in the knowledge-based context:

ContentEntity: Knowledge objects with specific content are shown or hidden to achieve dynamic personalization in this case. A distinction can be made between the cases where the complete knowledge object representing the content is shown or hidden or only a sub-element of such a knowledge object, that is, the granularity of the knowledge object content.

AssociationEntity: Associations or relations are shown or hidden to achieve personalization concerning relationships between knowledge objects. They may be dynamically inserted into a knowledge object to provide a knowledge worker with details, or more relevant information, or to cater to changes in short-term interests. In such association addition cases, care must be taken to prevent adding too many associations; otherwise it may lead to the “information overload” or “cognitive overload" phenomena. Similarly, associations may be dynamically removed from the knowledge object to reduce cluttering, and to allow the knowledge workers to concentrate on the current context, thus increasing efficiency of the knowledge worker.

StructureEntity: the entity that is personalized in this case is the structure of the entities. Knowledge objects may be structured in various ways, for example hierarchical tree structure, network, stars, pipeline, and so on. Dynamic personalization may take this structuring into account and show or hide only certain structures or structural composites depending on the context of the user, e.g. for a hierarchical structure, only some branch of the hierarchical tree may be shown to a knowledge worker depending on her current context.

Fig. 3 shows the basic entity PersonalizableEntity:

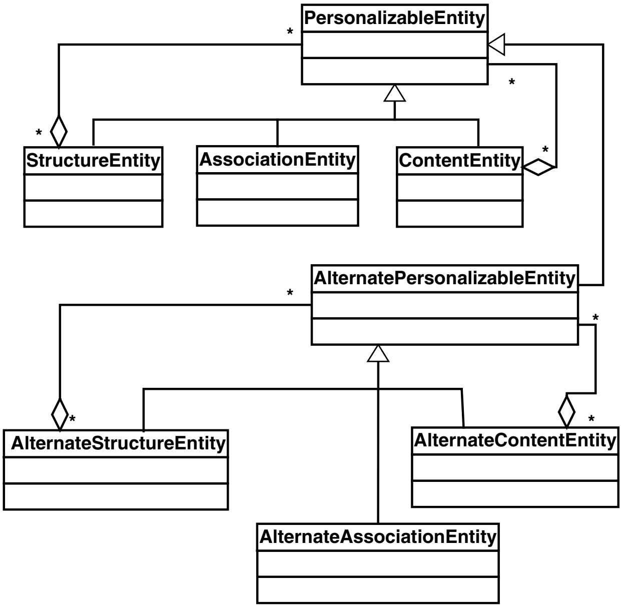


Fig. 3. PersonalizableEntity

AlternatePersonalizableEntity: An AlternatePersonalizableEntity inherits from the PersonalizableEntity, and represents an alternative presentation form. For example, in an eLearning environment, the concepts already learned might be marked in a different color, say green, from the text that is not yet ready to be learned, which may be marked in red. Such a representation is an example of AlternatePersonalizableEntity.

PersonalizationEngine: PersonalizationEngine is the entity that is responsible for performing dynamic personalization. Dynamic personalization is influenced by various kinds of events, for example, change of device, location, and network characteristics. Other events are either a result of explicit feedback by the user or implicit achievement of certain milestones; for example, a student on completion of certain topics and exercises gains expertise in that topic. The changes in UserContext trigger Event Condition Action (ECA) policies, which result in dynamic personalization of PersonalizableEntity. Similarly, user navigation or user-triggered events lead to update of UserContext and eventually to dynamic personalization as a result of evaluation of ECA policies.

The complete meta-model for dynamic personalization is shown in Fig. 4:

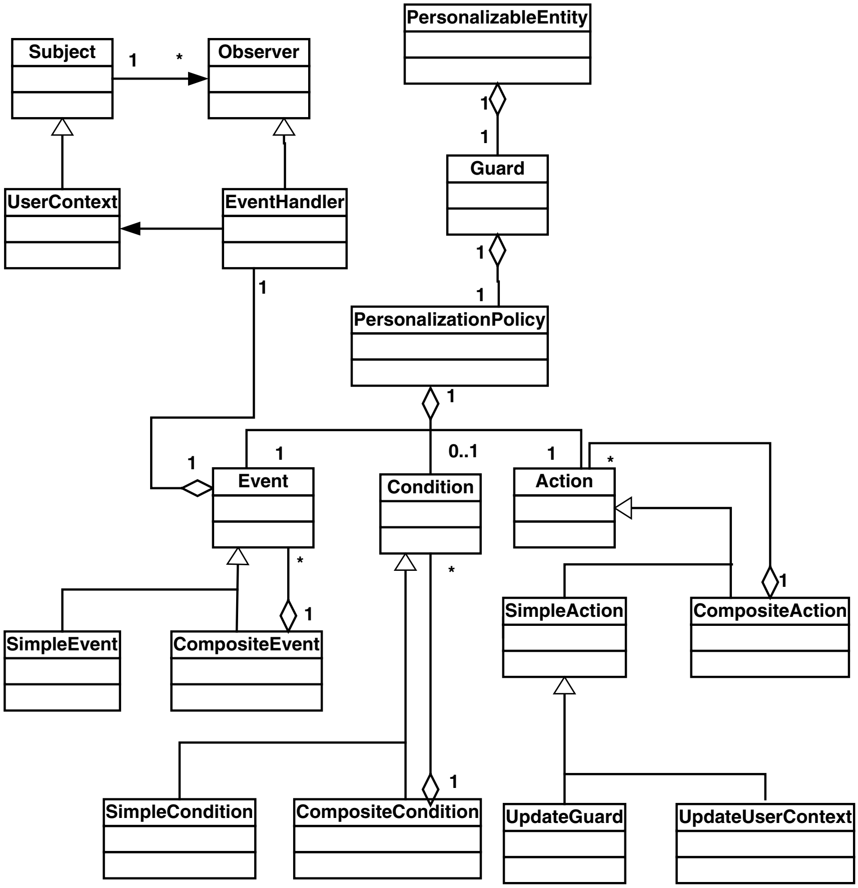


Fig. 4. Dynamic Personalization meta-model

Following are some of the features of the dynamic personalization meta-model:

- Each PersonalizableEntity or AlternatePersonalizableEntity has a Guard Component associated with it that activates or deactivates that particular entity. A Guard, once triggered by the PersonalizationEngine evaluates certain Boolean conditions that evaluate to True or False, and accordingly performs the activation or deactivation of the PersonalizableEntity or AlternatePersonalizableEntity that it is associated with.

- The changes related to UserContext are used to trigger the corresponding EventHandlers, which are part of the related Events. For example, a change in device used by a user to access the system will trigger the DeviceEventHandler that is part of the DeviceEvent.
- Each Guard has a PersonalizationPolicy, that is, an EventConditionAction rule (which consists of an Event, an optional Condition, and an Action) associated with it. An Event may be a SimpleEvent like a “TimeEvent” or a CompositeEvent consisting of one or more SimpleEvents related with AND, OR, and NOT. An example of CompositeEvent is “TimeEvent AND LocationEvent”. Similarly Condition may be a SimpleCondition or CompositeCondition consisting of one or more SimpleConditions related with AND, OR, NOT, LT, LTEQ, EQ, NOTEQ, GT and GTEQ. An Action may be a SimpleAction or a CompositeAction consisting of one or more SimpleActions. The SimpleAction causes the Guard to switch from active to inactive state or vice versa, and updates UpdateUserContext which in turn leads to the updating of UserContext, and thus to further dynamic personalization by evaluation of a new PersonalizationPolicy.

4 Architecture

This section introduces a generic architecture for systems based on the dynamic personalization meta-model and the domain-specific models.

The generic architecture is based on the multi-tier application architecture that provides the separation of concerns between the various tiers, and dedication of each tier to its own area of specialization. Such multi-tier systems typically consist of the following tiers:

- Presentation or Client tier: consisting mainly of the presentation logic.
- Middle tier or middleware: consist of one or more modules providing services and business logic to the application, and
- Data tier: consisting of databases and other data management capabilities.

The middle tier interfaces with the presentation and the data tiers to provide client services and data interface to the application.

Fig. 5 shows the Detailed Software Architecture of the Dynamic Personalization Engine which is an extension of the architecture presented in [8].

The Presentation Tier consists of multiple clients interacting with the PersonalizationEngine bi-directionally. This is a standard interaction, and will not be detailed any further.

The Data Tier consists of the databases PersonalizationPolicybase, UserContext, PersonalizableEntity, AlternatePersonalizableEntity, PersonalizableEntityMetadata and the AlternatePersonalizableEntityMetadata that store all the data that is relevant for the Dynamic Personalization Framework. The PersonalizationPolicyBase stores all the PersonalizationPolicies, which is, ECA (Event Condition Action) Rules. UserContext stores the context of the user, both environment and user profile related. PersonalizableEntity database stores the PersonalizableEntities. AlternatePersonalizableEntity databases store the AlternatePersonalizableEntities. PersonalizableEntity-Metadata stores the original metadata related to the PersonalizableEntities. Personal-

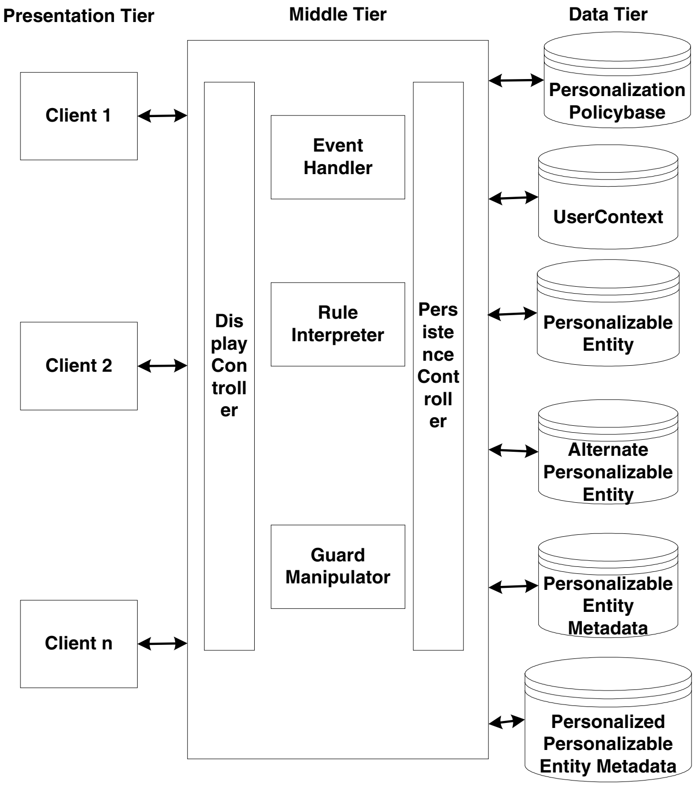


Fig. 5. Suggested architecture of a system providing dynamic personalization

izedPersonalizableEntityMetadata stores the metadata that have been personalized to suit a particular user's needs. PersonalizableEntityMetadata stores the original metadata that is not touched or modified in any way, whereas the PersonalizedPersonalizableEntityMetadata stores the changes or requirements of different users so that they are not required to input their metadata related preferences again and again.

The PersonalizationEngine constitutes the middle Tier, and consists of the modules DisplayController, PersistenceController, EventHandler, RuleInterpreter, and the GuardManipulator.

DisplayController and the PersistenceController form the interfaces of the Middle Tier to the Presentation and the Data Tiers respectively.

Changes caused either by the user directly or the user environment trigger the EventHandler.The EventHandler accesses the relevant PersonalizationPolicy from the PersonalizationPolicyBase, updates the UserContext database, and triggers the Rule Interpreter.

The Rule Interpreter assesses the PersonalizationPolicies by comparing the relevant entities from the UserContext databases and the PersonalizableEntityMetadata or the Alternate PersonalizableEntityMetadata depending on the PersonalizationPolicy. The action involves triggering the Guard Manipulator.

The GuardManipulator either activates or deactivates the relevant PersonalizableEntity or the AlternatePersonalizableEntity, and triggers the DisplayController, which performs the necessary presentation-related transformations. A second action performed by the Rule Interpreter, in response to the internal events may be updating of UserContext.

5 Personalizing Structure

The generic meta-model presented in chapter 3 is an important part of a comprehensive framework for dynamic personalization. The framework consists of a meta-model, an instruction for how to map the generic meta-model to domain specific models and a number of mappings of the meta-model to domain specific models. Among these domains are for example hypermedia, eLearning and Knowledge Management. In [5] we have examined how the generic meta-model incorporates all elements, operations and relationships supported by two hypermedia reference models: The Dexter Hypertext Reference Model [4] and the Adaptive Hypermedia Application Model AHAM [0]. The representation style has to be adapted to the individual requirements of knowledge workers and thus needs to be individually adapted in almost any single case.

5.1 The Structural Abstraction

With this paper we are especially concerned with the representation style of relationships between elements of the model. There has been research in the field of Structural Computing concerning what the *structural abstraction* is (for instance [5], [1] and [2]). In other words: What is the most atomic structural relationship allowing for building up high-level structural abstractions that are tailor-made for specific domains? In the case of our meta-model we consider the construct from Fig. 6 as most atomic structural abstraction of our meta-model.

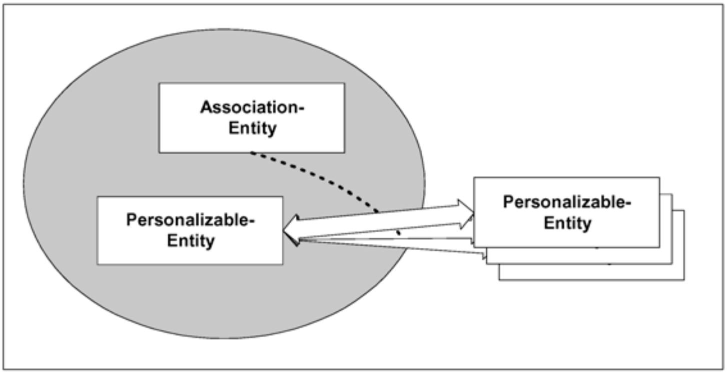


Fig. 6. The most basic structural abstraction for dynamic personalization according to the meta-model introduced in chapter 3

- A PersonalizableEntity is associated to an arbitrary number of entities of the type PersonalizableEntity.
- The association is provided by an instance of the type AssociationEntity.
- In object-oriented terminology, AssociationEntity is derived from PersonalizableEntity.
- A PersonalizableEntity can also be specialized in the form of an instance of the type ContentEntity.
- A PersonalizableEntity can also be specialized in the form of an instance of the type StructureEntity.
- ContentEntity as well as StructureEntity can contain an arbitrary number elements of the type PersonalizableEntity.
- On the most basic level the structural abstraction provided consists of an element of the type PersonalizableEntity and an AssociationEntity that 'points nowhere'[1].
- On the next higher level, the structural abstraction consists of two related elements of the type PersonalizableEntity where the relationship is provided by the AssociationEntity.
- Higher level structural abstractions consist of an arbitrary number of elements of the type PersonalizableEntity related to each other arbitrarily via a number of elements of type AssociationEntity.

Comparing the above said with the class diagrams from the generic meta-model (see chapter 3) a few issues appear. First, we believe that the structural atom needs to allow for unambiguously specifying the type of structure it provides. The StructureEntity is not intended to do so, but it serves to contain any type of structure, e.g. hierarchical tree-like structures, that are dynamically constructed during personalization.

Second, the generic meta-model does not provide a rigid relationship between content and relation. Seen from a structural viewpoint the role of the relationship needs to be elevated compared to the role of content. Therefore a rigid connection between PersonalizableEntity and AssociationEntity has to be provided allowing a structural model to perceive and treat content and relation as a unity.

From a structural viewpoint, the meta-model outlined above needs to be supplemented with the classes presented in Fig. 7. The classes' characteristics are as follows:

StructuralPersonalizableEntity: The StructuralPersonalizableEntity is derived from the original PersonalizableEntity and enhances it in two ways:

StructuralPersonalizableEntity has a rigid connection to StructuralAssociationEntity, which allows the pair to be only treated as a unity.

There is no separate element StructureEntity that provides mechanisms for building any kind of structure. All structuring functionality lies in the StructuralPersonalizableEntity.

StructuralContentEntity: This class needs to be provided instead of reusing ContentEntity. Contrary to ContentEntity, StructuralContentEntity allows to contain an arbitrary number of elements of the type StructuralPersonalizableEntity. Except from that, StructuralContentEntity does not specialize the behavior of ContentEntity.

[1] The semantics of the terms 'point' and 'nowhere' are described in more detail below.

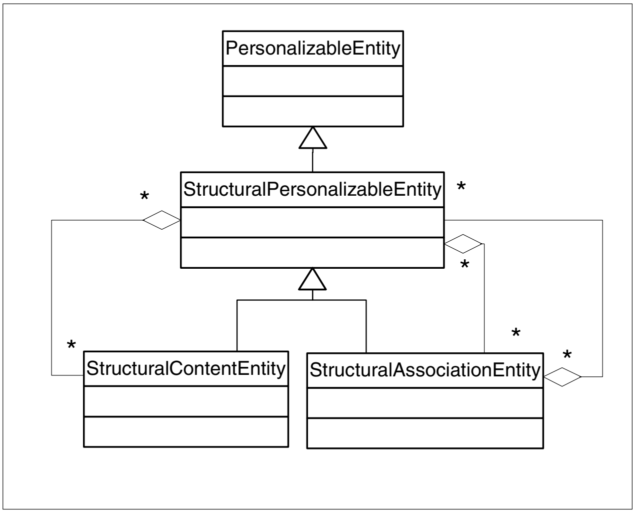


Fig. 7. Class structure that allows treating the basic structural abstraction as a unity

StructuralAssociationEntity: Elements of the type StructuralAssociationEntity specify the type of relationship that represents the structural abstraction. Their main characteristics are as follows:

- In any structure, the StructuralAssociationEntity provides the relationship between the more lower-level structural abstractions. The lowest level structural abstraction is a pair of StructuralPersonalizableEntity and StructuralAssociationEntity that does not contain any further element of type StructuralPersonalizableEntity (i.e. that 'points nowhere').
- StructuralAssociationEntity contains zero or more elements of the type StructuralPersonalizableEntity. Hence, to any relationship in a structure a type can be assigned.

We believe that our design allows us to deal with the following issue: The basic task of StructuralAssociationEntity is to assign types to relationships between two or more elements of the type StructuralPersonalizableEntity. In case we need to assign a type to a relationship between a StructuralPersonalizableEntity and StructuralAssociationEntity, our design allows doing that. Any association may contain a reference to an element of the type StructuralAssociationEntity, thus allowing assigning types to typed relationships (compare Fig. 8).

There are still a number of open questions concerning the proposed design. We might need to extend the given concept in the future to address the following issues:

- There are a couple of issues concerning scalability and performance of the proposed class model: According to the given model structure is built from combining recursively pairs of StructuralPersonalizableEntity and StructuralAssociationEntity. It is not clear yet, if this leads to data structures computers might not be able to handle.

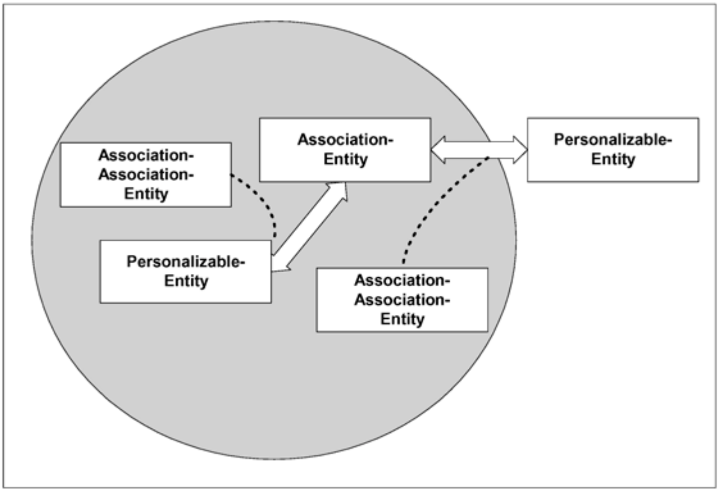

Association-Association-Entity
Association-Entity
Personalizable-Entity
Personalizable-Entity
Association-Association-Entity

Fig. 8. Assigning a type to the relationship between the PersonalizableEntity and the AssociationEntity with an 'AssociationAssociationEntity'

- There are questions concerning storing, maintaining and presenting the structural abstractions: At the moment the structural abstraction is still made up of two separate types of elements, one representing *data* the other one representing *structure*. This poses the question where to store and maintain these types of elements and when to combine and present them to clients? Is the middleware layer the right place to build the structural abstractions out of separate types of elements, would it be beneficial to store them as unity in the backend layer or shall we push computation concerning the structure into the operating system level?

These questions are not addressed or even answered yet and need to be dealt with in future research.

5.2 The Structural Transformation

In chapter 3 it is described how Guards, Events and Actions provide the system's dynamic capabilities. These elements allow for dynamically personalizing knowledge and for presenting it to clients in a way that is specifically adapted to that client. In this section we will examine whether the functionalities provided by these elements meet the requirements that arise from treating structure as first-class concept. We therefore examine the concept of structural transformations. According to the work of Anderson described in [1] and [2], structural abstractions need to be transformed due to events that might occur through intervention from external or internal sources. Two examples for these interventions outlined hereinafter:

In a hypermedia system, a user reads a section that she wishes to be explained in more detail. She clicks on the section and the original content is replaced by a more detailed version of the section, providing more information to the current topic. This is an example for an external event that causes structural transformation: The original structure has to be expanded and include the new element that is, somewhere in the

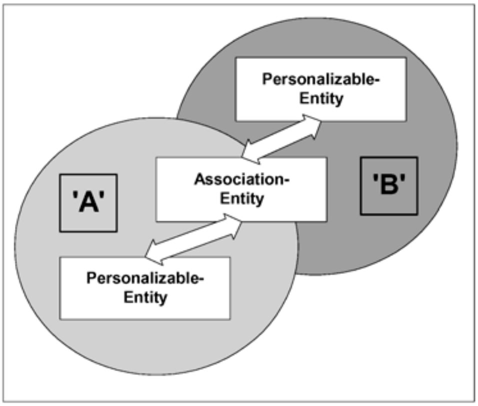


Fig. 9. Example for a structural transformation: Original state: Between two elements ‘A’ and ‘B’ of the type PersonalizableEntity a relationship is established

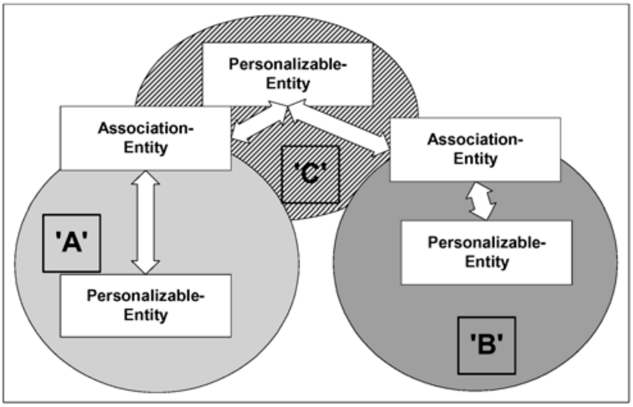


Fig. 10. Example for structural transformations: Final state: A new PersonalizableEntity ‘C’ is inserted into the existing structure

original structure a relationship between two elements has to be changed and an additional element is inserted (compare figures Fig. 9 and Fig. 10).

An example for an internal event is when a user decides to view a number of ‘information items’ sorted for how currently they have been created. The ‘information items’ are made up of one or more of the structural abstractions outlined above. The sort order representing the relevance is the highest level structural abstraction in the given context. In the case a newly created item is inserted it must be placed on top of the sort order. The structural transformation is triggered by an event that occurred from within the ‘system’[2].

[2] We are talking of ‘system’ and ‘information items’ in an abstract way. Both terms only serve the illustration of the example.

Both examples are taken from a taxonomy of adaptive hypermedia techniques listed by Brusilovsky in [3]. In the following table we show a number of hypermedia techniques from Brusilovsky's taxonomy and how the corresponding structural transformation might look like (the table is partly taken from [5]):

Table 1. Hypermedia techniques and corresponding structural transformations

Hypermedia technique	**Description**	**Structural transformation**
Inserting / removing fragments	Fragments (information items) are added or removed.	The relationship between two elements of type StructuralPersonalizableEntity is changed: The StructuralAssociationEntity points to a different StructuralPersonalizableEntity than before. (compare figures Fig. 9 and Fig. 10)
Altering fragments	The level of detail of a fragment is altered	A fragment is removed and an alternative fragment is inserted (see above for inserting and removing fragments)
Stretchtext	Clicking on a specific region in a fragment leads to replacement of original region by alternative fragment	A fragment is removed and an alternative is inserted (see above)
Sorting fragments	Fragments are sorted according to specific sort order	A number of remove / insert operations according to sort criteria (see above)
Disabling link	The anchor does not activate the link	The StructuralAssociationEntity assigned to the relationship (link) does not point to a StructuralPersonalizableEntity
Removal of link	The anchor and link are removed	Same as link disabling. Additionally, the StructuralContentEntity is advised not to display the link any more
Adaptive link sorting	Links are sorted according to specific sort order	A number of remove / insert operations (see above)
Adaptive link	Adding comments (types) to links	Assigning an element of type StructuralContentEntity to the StructuralAssociationEntity
Adaptive link generation	Non-authored links are generated	System examines StructuralContentEntity, retrieves all references and creates and inserts elements of type StructuralAssociationEntity

There are further techniques described in Brusilovsky's taxonomy. [3] and [5] provide a more thorough description of theses techniques. The structural transformations are triggered by activation or deactivation of Guard elements according to the specific PersonalizationPolicy used (see chapter 3). The concepts introduced in chapter 3 appear to apply to the examples we provided in Table 1. We conclude from that that the structural abstractions we derived from our generic meta-model is applicable to at least one example domain.

6 Conclusions

With this paper we have shown our results from examining a meta-model and architecture from a perspective that takes structure rather than data as main focus of interest. It has been shown that it was sufficient to derive three new classes from the original class model in order to construct structural abstractions that serve as basic conceptual entities of our model. With that we augmented the model in a ways that makes structure, as opposed to data, the first-class concept. We have additionally shown that the behavior of a system that supports dynamic personalization of knowledge might also be realized. Our findings, at least, show that this conclusion applies to the example domain adaptive hypermedia. In the future there needs to be more thorough examinations concerning the following issues:

- System architecture: We did not examine in detail yet how the requirements of Structural Computing influence the proposed system architecture. Open issues are, among others, where to place the processing of the structural abstractions: Most of the processing in our architecture is currently being done in the middle-tier. In future work we will certainly need to augment the data-tier with the ability to store and process structural abstractions.
- Structural transformation: We did not include examinations of generic structural transformations in this paper. Instead we provided an examination of one specific example domain: adaptive hypermedia. Hence, in the future we will have to analytically develop more generic structural transformations for dynamic personalization.
- Example domains: It is planned that in the future, the generic meta-model will be mapped to a number of example domains such as eLearning, eCommerce or Knowledge Management. In parallel we will investigate whether the specific mappings can be comprehended and reproduced in the structural domain, i.e. whether or not the structural abstraction and transformations suffice to dynamically personalize knowledge in a number of diverse domains.

Acknowledgements

The Know-Center is a Competence Center funded within the Austrian Competence Center program K plus under the auspices of the Austrian Ministry of Transport, Innovation and Technology (www.kplus.at).

References

1. Anderson, M.K.: Structural Computing Requirements for the Transformation of Structures and Behaviors. Lecture Notes in Computer Science, Vol. 1903. Springer-Verlag, Berlin Heidelberg New York (2000) 140-146
2. Anderson, M.K., Sherba, S.A.: Using Structural Computing to Support Information Integration. Lecture Notes in Computer Science, Vol. 2266. Springer-Verlag, Berlin Heidelberg New York (2001) 151-159

3. Brusilovsky, P.: Adaptive Hypermedia. User Modeling and User-Adapted Interaction, Vol. 11, Kluwer Academic Publishers, Dordrecht, Netherlands. (2001). 87-110
4. Halasz, F., Schwartz, M.: The Dexter Hypertext Reference Model. In Communications of the ACM, Vol. 37, Nr. 2, 1994. (1994) 30-39
5. Kandpal, D., Ulbrich, A., Tochtermann, K.: Augmenting knowledge-based systems with dynamic personalization concepts. In Proceedings der 4. Oldenburger Fachtagung Wissensmanagement, Oldenburg, Germany (2003).
6. Nürnberg, P.: Repositioning Structural Computing. Lecture Notes in Computer Science, Vol. 1903. Springer-Verlag, Berlin Heidelberg New York (2000) 179-183
7. Tekinerdoğan, B., M. Saeki, G. Sunyé, P. van den Broek, and P. Hruby: "Automating Object-Oriented Software Development Methods". In Á. Frohner (Ed.): ECOOP 2001 Workshops, LNCS 2323, 2002, Springer-Verlag Berlin Heidelberg 2002, 41–56
8. Tochtermann, K.: Personalization in the context of digital libraries and knowledge management. Post-Doctoral Thesis, Graz University of Technology (2002).
9. Tochtermann, K.: Personalization in Knowledge Management. Lecture Notes in Computer Science, Vol. 2641. Springer-Verlag, Berlin Heidelberg New York (2002) 29-41
10. Wu, H., De Kort, E., and De Bra, P.: Design Issues for General-Purpose Adaptive Hypermedia Systems. In Proceedings of the ACM Conference on Hypertext and Hypermedia, Århus, Denmark. (2001) 141-150

Some Notes on Behavior in Structural Computing

Michalis Vaitis[1], Manolis Tzagarakis[2, 3],
Konstantinos Grivas[3], and Eleftherios Chrysochoos[3]

[1] University of the Aegean, Department of Geography
University Hill, GR-811 00 Mytilene, Greece
vaitis@aegean.gr
[2] Computer Technology Institute
Riga Ferraiou 61, GR-262 21 Patras, Greece
tzagara@cti.gr
[3] University of Partras, Computer Engineering and Informatics Department
GR-265 00 Patras, Greece
{grivas,chrisoch}@ceid.upatras.gr

Abstract. Behavior has only recently received the attention of structural computing research. Thus, the exact relation of behavior to structure is to a large extent unspecified. Current structural computing research still views behavior as something that is orthogonal to structure: something that has to be added on top of it. In this paper, we attempt to create a framework to discuss the notions of *structure*, *function* and *behavior*. We argue that a *structure* exhibits *behavior*, behaviors effects *function*, and function enables *purpose*. Moreover, we point out *propagation* as an inherent behavioral capability of relationships. This discussion aims at providing the ground to recognize the variant and invariant behaviors found in structures.

1 Introduction

Structural computing is a research area driven by the necessity to generalize the "node-link" paradigm in navigational hypertext, and at the same time to offer an infrastructure for the specialization of structuring semantics, according to user or application needs. Based on the philosophy of the "primacy of structure over data", the aim of structural computing is to study the notion of structure and develop systems offering structural abstractions, rather than data abstractions. Although the term "structural computing" first appeared in 1997 [7] and three workshops have already been held on the subject [5, 8, 9], fundamental questions are still unanswered. This is especially true when considering the notion of behavior in structural computing.

Previous work on structural computing has emphasized on the fulfillment of syntactic requirements, such as the valence, scope and directionality [10] (i.e. "what and how it can be constructed"). Building on these concepts, structure types and templates [14], or fundamental association types [4] have been proposed attempting to elevate structure from an implicit property of data elements, to an explicit definition of relationship semantics. Nevertheless, while such constructs consist primarily syntactic entities, focusing on the static aspects of a problem domain, they do not answer the issues of how they *behave*. The role of behaviors in hypermedia has been pointed out

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 143–149, 2004.

many times, e.g., [12], and led to their explicit representation within the system. Consequently, the behavior of a structure is currently considered as an “add-on” that is developed on top of it, usually carried out after the definition of the syntactic laws. However, the question that arises is whether such manichaistic conceptualization of the problem (structure vs. behavior) provides the fruitful ground for further research in structural computing, given that some researchers recently review equivalent problem declarations, favoring for a viewpoint rather than axiomatic formulation[1].

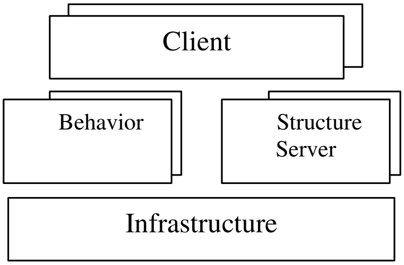


Fig. 1. CB-OHSs architecture

Until now, structural computing has succeeded in providing the evidence to acknowledge a behavioral component within the architecture of CB-OHSs (thus abstracting behaviors), but neglected to elucidate its precise role as well as its relationships with other entities (fig. 1). The behavioral component is usually treated as a “black box” that, under some (rather fuzzy) *circumstances*, interacts/operates upon structural abstractions in order to achieve a *goal*. Under which circumstances behaviors are activated and what goal is to be achieved, are still left unspecified. In our view, the inability to answer precisely such questions can be traced down to the (rather curious) fact that there are no attempts to define even the most commonly used terms in structural computing, such as “structure”.

Considering the term “structure” from a purely syntactical point of view, proves unable to compare and discuss systems and models within hypermedia. For example, while the two designs of figure 2 look homeomorphic (i.e. they exhibit a topological equivalence), they are radically different in the way they are experienced, thus admitting a behavioral differentiation.

While the UML class diagram (a) depicts a (fairly trivial) navigational model (where links can be endpoints of links, as it is the case in Chimera [1]), design (b) depicts the class diagram of a taxonomic model. Comparing both diagrams at the syntactic level would lead in supporting – within a structural computing system – a very abstract design, such that depicted in (c). However, such a design would be unable to reproduce the models from which it originated, since during the reduction step important information were lost: namely, the fact that the designs (a) and (b) are very different in their *purpose (goal)*. More clearly, seeing the designs (a) and (b) as form producing mechanisms (or patterns), they produce different forms when the same input is present. It is exactly this *form producing capability* that was lost during the reduction step. By form producing capabilities we mean that the above designs map a

[1] Personal communication with Peter Nuernberg, on the “data vs. structure vs. behavior” question.

particular input to different outputs, although their internal configuration is exactly the same.

In this paper, we attempt to create a framework to discuss the notions of *structure*, *function* and *behavior*. These notions are not orthogonal to each other, but are the different viewpoints of the same set of "things", which all together form a consistent *whole*; a whole that the end-user experiences as *structure*. A structure exhibits *behavior*, behaviors effects *function*, and function enables *purpose*. Purpose is directly related and originates from the various hypermedia domains. The framework aims at identifying the variant and invariant behaviors of a domain, which is a very important aspect when designing and developing structural computing systems.

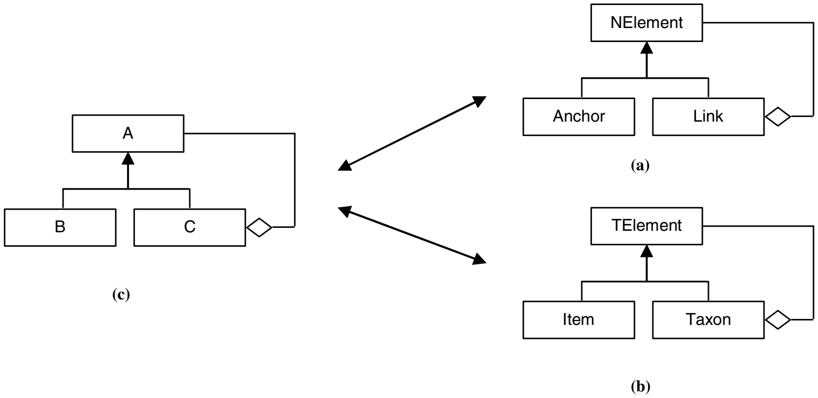


Fig. 2. UML class diagrams: (a) navigational domain, (b) taxonomic domain, (c) abstract design

2 Positioning the Concepts of Structure, Function and Behavior

Our view about structure is motivated by the fact that the way a structure is constructed, is always depended on the problem to be solved. Thus, structures are always the solution to a problem. Since within the problem statement and analysis (i.e. the hypermedia domain analysis) *patterns of activities* are outlined [6], it seems that when referring the term structure, a particular behavior is already assumed. Thus, structure and behavior appear tightly coupled and – at least from a user's point of view – undivided. Formulated differently, structure and behavior are different views of the same *whole*. Proper structures exhibit *fitness* with respect to a particular target problem domain.

Although a definition of the term structure is given below, at this point we realize structure as a set of interconnected elements. These interconnections form a configuration and the set of valid configurations (as it originates from the problem domain and encoded in the syntactic constraints) establishes the set of potential *states* the structure may have. The transition from one state to another corresponds to *changes* of the elements' configuration and thus to a *transformation of the structure*. Such changes are initiated by the execution of operations upon the structure. For example, printing a taxonomy means that the *print* operation is applied on a taxonomic structure, whereas deleting a link would mean the application of the *delete* operation upon the link abstraction within a navigational structure.

In most systems and models, the operations are regarded as functions that map an input state to an output state. Operations (and the algorithms they reify) reside usually "outside" the structure. These operations contain all the necessary syntactic and semantic constraints of manipulating structures. Therefore, a structure is the *object* of an execution (i.e. an algorithm). As a result, the semantics of the structure are not clear, but are buried within the algorithm.

Since structural computing is envisioned as the framework to express "primacy of structure", this approach seems questionable. Structure being an *object* and never *subject* of an operation or algorithm, raises concerns with respect to the term "primacy". A "primacy of structure" would also mean that *control* is on the *structure-side* and not (exclusively) on the *operation-side*. This means that, rather than *handing a structure to an operation/algorithm* to operate upon it, *an operation/algorithm is given to the structure to be executed*. This way, control is inverted: structure becomes the subject and not the object of an execution (algorithm). It is within the scope of this paper to discuss the terms structure and behavior within a *computational object/subject* or *caller/calle* dialectic. Primacy certainly has to do with control issues.

The approach stated before, elevates structure to the level of function. Thus, *structures function* in the sense that *they map inputs to particular outputs*. By the term *input* we denote a set of stimulus and structures, whereas by the term *output* we denote a new structure (i.e. a new configuration) and new stimuli. In this sense, it is said that a *structure transforms* or *reaches a new state* (fig. 3).

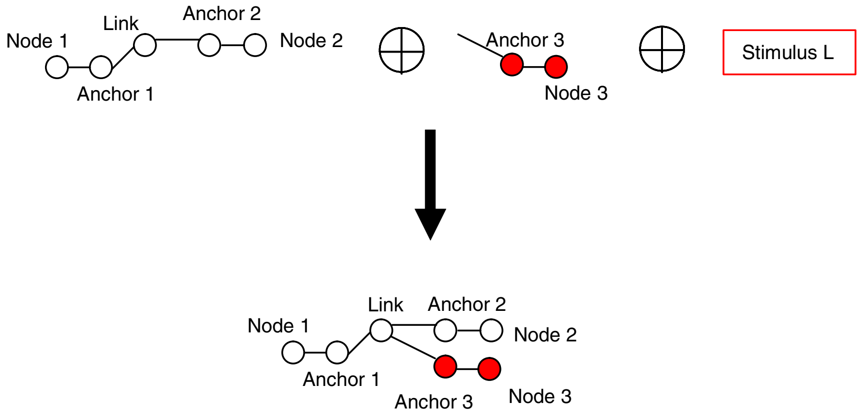


Fig. 3. Structure transformation

The relationship among structure and function – although not explicitly – has been already pointed out. In particular, in [6] the problem of representing odd numbers is mentioned. Odd numbers can be represented as a set {1, 3, 5, …}, where an explicit mapping between two consecutive numbers exists (called *next_odd_number*, so that *next_odd_number(1) = 3*). They can also be expressed by the formula (2*n)+1 that represents a mechanism producing odd numbers. Both morphisms can be considered structures and both are at the same time function or behavior.

Since structures function, the question that arises is how exactly such functioning (i.e. mapping) occurs. That functioning is not happening "instantly" or "quickly". Instead, it happens gradually and in stages. The acceptable states a structure undergoes, as a result of an external stimulus, in order to be able to fulfill a particular mapping and thus to create a specific output (i.e. reach a new state), is called the *behavior of the structure*. To make the difference between *function* and *behavior* clearer, we have to notice that the distinction between function and behavior is a viewpoint issue: if one looks only at the initial and final state of a structure, a mapping or transformation is perceived as a *function*, whereas if one witnesses the development of that transformation in time (all the distinct stages it takes until it reaches its final stage) will perceive it as *behavior*. Considering the aforementioned example of odd numbers, the formula (2*n)+1 can be seen as the behavior of the mapping called *next_odd_number* as introduced by the set {1, 3, 5, …}[2]. From the definition of behaviors it is obvious that each step – taken independently in the chain of state changes – is a transformation and thus a mapping or function. Since each such behavioral step is a transformation, it produces a new state and stimuli (as mentioned earlier). Given the fact that the acceptable states a structure undergoes are ordered, it follows that their ordering can only be justified on the basis of the causal connection between two consecutive steps: that the output of one transformational step is the input of the other. That way, the overall *behavior and function of the structure occurs*, meaning that at this point structures are *self-maintaining constructs*; self-maintaining in the sense that are able to hold the loads accepted or generated by them. These observations lead to the recognition of *stimuli propagation phenomena* within structures.

As *structure* we define a particular configuration of elements that may hold a load. A structure *holds a load* if it produces states (i.e. transforms in a particular way), which are consistent with the target problem domain. Structures that do produce states that are not consistent with the particular problem domain are said to *collapse* or to *break*. The term consistency here is referring to the fact that the produced structures can be observed in the problem domain.

3 Propagation: A Capability of Relationships

The question of how to perceive structures, so that they constitute self-maintaining constructs, can be addressed when considering a rather unnoticed capability of relationships: they manifest *interactions between the connected elements* and not only their *existential dependencies*. Interaction between elements simply means that an operation applied to an element has implications (or effects) to the interconnected element(s). For example, deleting a navigational node has the effect that all anchors contained within the node are also deleted. Such propagation effects can be witnessed in structures and – as it has already been pointed out – consist a characteristic of relationships [2, 11].

By exploiting the propagation capabilities of relationships, important issues may be explained, such as how localized behavioral phenomena (such as clicking an anchor, or changing a single attribute of a node) that are not visible to the end-user (in terms

[2] It is worth noticing that in the formula *(2*n)+1*, *n* denotes the *distance* from the initial state, i.e. 1.

of their mechanics), lead to global systemic phenomena (such as traversing a link) that the user is foremost experiencing [10].

Within the Callimachus framework, *propagation* is an inherent function/behavior of the structural elements. It manifests a kind of generic mapping: what operation will be applied to the elements of an endset, given that the elements of another endset (of the same structural element) have been received an operation. Propagation constitutes an event, which explicitly takes into consideration the references within the endsets of a structural element [14]. Currently, the possibility of recognizing *propagation patterns* within structures is investigated.

4 Related Work

In the object-oriented paradigm the notions of structure, function and behavior are strongly related. The universe of discourse is divided in object classes, each one having a set of internal properties (structure/state) and a set of methods (behavior), acting upon the state. The methods that are declared as "public" constitute the object's interface to the rest of the world (environment). Encapsulation, inheritance and polymorphism are considered the most remarkable characteristics of the object-oriented paradigm, offering a number of benefits in the areas of conceptual modeling, software designing and implementation.

Although its wide acceptance, the object-oriented paradigm has received some criticism about the modeling capabilities it offers (e.g., [13]). For example, no direct support exists for the handling of relationships among objects. In contrast, structure is manifested by the relationships – and their properties – that hold among a set of elements.

Also, in real world there are tasks concerning many different objects, where it is not clear which class is appropriate to be assigned with the corresponding methods. This is especially true for structural computing, as a structure is usually greater that the sum of its elements. Some behaviors are performed by the whole (not tangible) structure, rather than by anyone of its elements.

In addition, in the object-oriented paradigm much attention is paid about the objects of a problem domain, while the subjects are under-emphasized. In structural computing, structures have control on how they operate; they do not only respond to external messages.

For the above reasons we argue that the object-oriented paradigm is inadequate to serve as a framework for structural computing.

Graph grammars on the other hand offer formal methods for the specification of both complex object structures and graph transformations (behavior). Of especially interest is the related "graph rewriting" methodology, where a sub-graph of a host graph is replaced by another sub-graph [3]. Further investigation is needed to deduce if it can be used for the formal specification of behavioral aspects in structural computing.

5 Conclusions

The cornerstone result from the above discussion is that the concept of structure includes both the syntactic rules for the configuration of a number of elements, and the functionality of its elements is order to fulfill its purpose ("hold a load").

The functionality of a structure is analyzed in a number of transformations (or mappings) that describe all the valid states the structure can take, according the semantics of the application domain. Parameter structures and stimulus are both the input and output for each transformation. An initial stimulus propagates through the structure elements, triggering a series of transformations, until the final state of the structure to be reached ("equilibrium"). Consequently, each element of the structure should have the knowledge of how to "produce" output, given an input stimulus.

These preliminary notes about behavior should be further clarified in order to be able to formalize an algebra or language, which is able to describe both the syntactic and the behavioral aspects of structure as a whole.

References

1. Anderson, K. M., Taylor, R. N., Whitehead, Jr. E. J.: Chimera: Hypertext for Heterogeneous Software Environments. Proceedings of the ACM European Conference on Hypermedia Technology, Edinburgh, Scotland (1994) 94-107
2. Bernstein, P. A.: Repositories and object oriented databases. ACM SIGMOD Record, 27 (1) (1998) 88-96
3. Blostein, D., Fahmy, H., Grbavec, A.: Practical Use of Graph Rewriting. Technical Report No. 95-373, Dept. of Computing and Information Science, Queen's University, Kingston, Ontario, Canada (1995)
4. Millard, D. E., Moreau, L., Davis, H. C., Reich, S.: FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability between Hypertext Domains. Proceedings of the Eleventh ACM Conference on Hypertext and Hypermedia, San Antonio, Texas, USA (2000) 93-102
5. Nürnberg, P. J. (ed.): Proceedings of the First International Workshop on Structural Computing, Technical Report AUE-CS-99-04, Aalborg University Esbjerg, Denmark (1999)
6. Nürnberg, P. J., Leggett, J. J.: A Vision for Open Hypermedia Systems. Journal of Digital Information, Special Issue on Open Hypermedia Systems, 1 (2) (1997)
7. Nürnberg, P. J., Leggett, J. J., Schneider, E. R.: As We Should Have Thought. Proceedings of the Eighth ACM Conference on Hypertext, Southampton, UK (1997) 96-101
8. Reich, S., Anderson, K. (eds.): Open Hypermedia Systems and Structural Computing. Proceedings of the Second International Workshop on Structural Computing, San Antonio, Texas, USA, LNCS 1903, Springer-Verlag (2000)
9. Reich, S., Tzagarakis, M., De Bra, P. (eds.): Hypermedia: Openness, Structural Awareness and Adaptivity. Proceedings of the Third International Workshop on Structural Computing, Aarhus, Denmark, LNCS 2266, Springer Verlag (2001)
10. Rosenberg, J.: A Hypertextuality of Arbitrary Structure: A Writer's Point of View. Proceedings of the First International Workshop on Structural Computing (1999) 3-10
11. Rumbaugh, J.: Controlling Propagation of Operations using Attributes on Relations. Proceedings of the ACM Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA), San Diego, California, USA (1988) 285-296
12. Schneider, E.: Link Behavior Diversity in the Open Hypermedia Systems Context. Proceedings of the Third Workshop on Open Hypermedia Systems, Southampton, UK (1997)
13. Shajan, M..: Critique of the Object Oriented Paradigm: Beyond Object-Orientation. http://members.aol.com/shaz7862/critique.htm (1997)
14. Tzagarakis, M., Avramidis, D., Kyriakopoulou, M., Schraefel, M., Vaitis, M., Christodoulakis, D.: Structuring Primitives in the Callimachus Component-Based Open Hypermedia System. Journal of Network and Computer Applications, 26 (1) (2003) 139-162

Strategies for Hypermedia Design Modeling

Ahmet Sıkıcı and N.Yasemin Topaloğlu

Department of Computer Engineering
Ege University, Izmir, Turkey
ahmets@staff.ege.edu.tr
yasemin@bornova.ege.edu.tr

Abstract. The conventional tools and methods that are used widely in software design are not adequate for expressing the complexity and flexibility of hypermedia artifacts. Visual, distributed, interconnected and knowledge dense character of hypermedia environments, require a revision over the traditional software design tools and concepts. In this work we will specify some aspects of hypermedia that are hard to be modeled with the state of the art approaches and propose some strategies that can help to overcome the problems.

1 Introduction

Hypermedia is a postmodern medium for the presentation of knowledge which has come into our homes with the popularity of the World Wide Web. It is also a knowledge representation tool that resembles the semantic topology of the information that it contains. Its power comes from the parallelism with the human way of thinking and the ability to stimulate intuition through some kind of realism effect. The psychological impact of this descriptive power makes hypermedia a valuable tool for research and education [13] [10].

The modeling approach to dealing with the complexity of a subject involves restricting or filtering it into something more manageable. However when this reduction is exaggerated, the essence of the subject may be lost. It is our opinion that the flexibility of the current object oriented approaches that are used to model hypermedia designs, are not anything near the flexibility of the hypermedia itself and this rigidity diminishes the expressiveness of the products.

In this work we will explore some of the properties of hypermedia design that are difficult to model with conventional approaches and we will propose solutions that aim a direct representation of design knowledge, instead of restricting it to a predefined shape. In this work some modern design and programming concepts like modularity, containment and encapsulation will be reconsidered with the aim of increasing the generality and flexibility of the languages of design.

The organization of the paper is as follows: In Section 2 the existing approach to hypermedia design is summarized with a very little discussion about the problems being faced. In Section 3, after a short introduction of the postmodern approach to production, some difficulties of hypermedia design are handled one by one and new strategies are proposed. Section 4 contains the conclusions.

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 150–157, 2004.

2 The Modern Approach to Structure

The modern answer to the software complexity relies heavily on the old *divide and conquer* rule. Each component of the product is made by its own specialized process and experts, and later the whole thing is assembled on a central *production line*. This approach is known as Fordism, or Taylorism and much of the software development has been grounded in this view of human industrial activity [9].

Taylorism has strong influences on the characteristics of the designs since it requires the target product to be decomposable into independent units that are called modules. This is called modular design and a modular design targets a modular architecture. A system with such an architecture functions as a set of modules accomplish their own tasks and interact through a minimal channel and language. The concrete benefit of this approach is that it is possible to replace a module with a new one, without interrupting the rest of the system as long as the *overall behavior* of the new module is compatible with the old one. This is what happens when a device breaks down and is fixed simply by replacing a chip or a circuit board with a new one.

With these concerns, modernism puts emphasis on standardization, mass production and the assembly of modules around a central design.

In the object oriented paradigm which is a faithful follower of the modern approach, this *overall behavior* is represented by the *interface* concept.

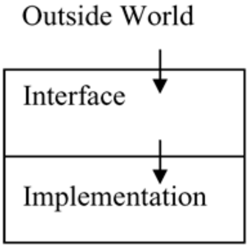


Fig. 1. Layering in an object

As seen in Fig.1. objects employ a local layering in order to prevent the outside world from accessing the inner privacy which is called *implementation*. This *micro layering* method is called *encapsulation*. When a piece of architecture is *encapsulated*, it is only possible to reach it indirectly through an interface. Thus the interface of an *encapsulated* module can be thought of as a *summary* of that module, which is the only part that the high level designer should care about. According to Cox [4] encapsulated complexity is no longer complexity at all. It's gone, buried forever in somebody else's problem.

The modern approach to hypermedia design is best represented by the Web Modeling Languages (WMLs). Some of the popular WMLs are OOHDM[12], WebML[2] and Conallen[3]. WMLs employ object oriented design, XML and some specialized models in various proportions. WMLs target the issues of conceptual design, arrangement of content, navigation, user interface and transactions. The general approach is to arrange these requirements as phases or layers of design and to handle each one with its own logic, tools and diagrams. However the integration of the solutions is not straightforward. There are usually problems in gluing hypermedia structure with software application logic [8]. And the reason is as Gomez et al [6] states it: The existing tools for building and deploying complex Web sites are inadequate for dealing with the software production processes that involve connecting with underlying logic in a uniform and systematic way. The literature indicates that there is incompatibility between hypermedia concerns and object oriented point of view.

3 The Postmodern Hypermedia

The postmodern thought has emerged as an opposition to the ideals of the modern school. Postmodern style of production praises heterogeneity instead of standardization, distribution of control instead of central control and flexible production instead of mass production. These require establishing stronger communication bonds between the parts of a system and having a holistic view about design. Factories that can be rearranged for new products or new capacity requirements, just in time delivery of materials, flexible work hours and custom-made industrial goods are examples of the post modern way of manufacturing.

However the most postmodern organization on this planet is most probably the World Wide Web. Starting from the underlying network architecture, up to the social patterns that it supports, Web is a perfect example of heterogeneity, dynamism, flexibility and distributedness. The problem with modeling the Web or any similar hypermedium with conventional tools is that modeling a postmodern artifact with a tool that belongs to the modern age is not feasible.

In this section we will try to demonstrate this infeasibility and propose new approaches to solve the modeling problems. Our target will be to gather the elements of a design language that is complete over a reasonable hypermedia-related design space. However, due to the newness of the subject and high potential of hypermedia in terms of variability, such a design space is not currently known but can be clarified implicitly as the language is being used. Therefore instead of checking for completeness over a domain, completeness can be preserved de-facto as the domain and the language evolve together.

3.1 Modularity Revised

Modularity is the main concept that characterizes modern design. However in the hypermedia world, sometimes it can be unnatural to structure certain reusable designs as modules.

The most fundamental feature of hypermedia that resists modularity is that hypermedia constructs can intersect or overlap physically like Siamese twins. This happens when a part of a construct is also contained by another construct in an ad hoc manner like referencing, or when two constructs share a common resource. Web is a place where a lot of overlapping takes place, mainly for practical or performance related concerns. First of all at the bottom level, HTML supports overlapping by allowing expressions like the following:

```
<b><small></b></small>
```

At a higher level there are a lot of audio-visual media, mostly illustrations that are shared among the pages. There are also a lot of pages, frames, banners and applets in the common use of multiple sites. At the topmost level we see that sites also can be shared by multiple *virtual cities* or *web rings*.

There is no perfect solution to the intersection problem. Or at least no solution can make the situation look as neat as in the case with pure modularity. However, overlapping is a fact in the hypermedia world and it demands serious attention. Instead of trying to ignore or avoid the intersection of the artifacts, intersection should be conceptually integrated into the model. This can be possible with a new point of view in which conventional modules are replaced by structures that can intersect but do not get lost by blending into each other or the environment. In that case the division of concerns can rely on precise definitions of each of the two intersecting structures as there is no other way to tell where one object ends and the other had started. We can still talk about modularity but this time on a conceptual level, instead of implementation level.

The design model for hypermedia should be capable of defining how an overlapping structure can be created, destroyed and recognized. It would also be very helpful to support a *superposition operation* that merges two structures into one. The Pattern Paradigm deserves attention as it supports such unification of constructs. With the words of Christopher Alexander [1] who is the founder of the Pattern Paradigm: It is also possible to put patterns together in such a way that many patterns overlap in the same physical space; the building is very dense; it has many meanings captured in a small space; and through this density, it becomes profound.

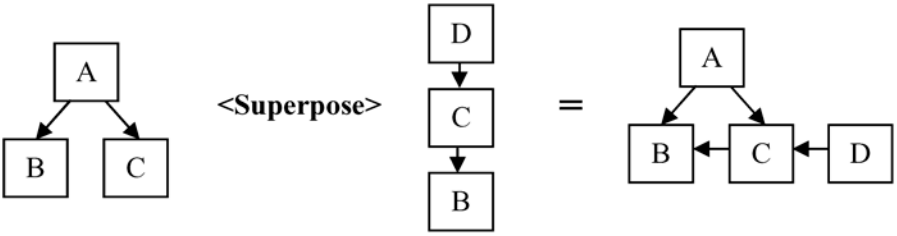


Fig. 2. Superposition

Fig.2. depicts superposition on two basic graphs. The operation preserves the structures of the two operands but compresses the overall knowledge into a smaller structure.

Another threat to the modularity of hypermedia systems originates from the high semantic coupling between the building blocks. The interconnectedness of hypermedia brings about the fact that the identity and meaning of an entity is highly dependant on the context in which it appears. The best examples of *context dependency* are found on the Web again. Web works become enmeshed in a context that reflects their meaning, use and construction [5].

In terms of Web design it may not be enough to know that a menu exists on the visual layout of a page. Due to the knowledge representation role of the hypermedia, the issues of design usually need to cover things like where the menu is located, where it provides navigation to and whether the menu exists on every page of the site or not. According to the context in which it appears the menu becomes something that makes sense for Web design like a top menu, a site navigation menu, main menu or any combination of these.

This problem should be addressed with a knowledge representation point of view. A design is meant to be semantically complete so it naturally should contain every piece of knowledge that contributes to its essence. Therefore a design artifact should be defined together with the context that it requires. However the format of the design should clearly indicate which parts belong to the context and which parts belong to the content.

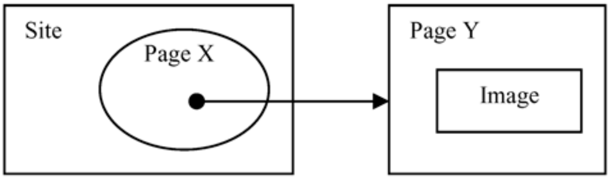


Fig. 3. Defining with content and context

In Fig.3. such a definition is basically exemplified. This design describes a Web page that possesses a navigational link to an external, illustrated page. The subject of definition is represented by the ellipse and labeled as *Page X*. The inside of this ellipse represents the content side of the definition and the outside of it stands for context. The content definition is simple: Page X has a navigation link. The context definition does the rest of the job and specifies that the target of this link is outside the site (that page X is in) and that this target page (called Y) possesses an image in it.

3.2 Containment Modeling

In order to perform an effective modeling of hypermedia we should standardize the definition of containment or at least we should have a consistent perspective of the concept. In [7] Gordon and Whitehead split containment relation into two types as *inclusive containment* and *referential containment*. The distinction between the two has been given in terms of memory allocation. In this work we are interested in the semantic meaning of containment rather than its implementation. In fact referencing an object is semantically quite different from containing it however in highly distributed environments like the Web, the distinction between containment and referencing gets blurry.

There are other semantic ways to classify a containment relation like whether the container is simply a bag that holds the containees inside or is it made up of them. In [10] Odell addresses the latter type under the name *composition* and defines six *composition types* in a semi-formal fashion, by alternating the combinations of three independent semantic aspects. However, although the elegance of the analytical approach of this classification is attractive, there is an inevitable vagueness inherent from the employment of natural language both in the problems and the solutions.

The exact definition of a containment relation seems to be relative to what is aimed to be modeled with it. However it can be possible to reach a meta-model through the concept of encapsulation. If on a given design space *S*, there is no way to associate with an entity *A* without interfering with an entity *B*, then we can say that *B* contains *A* according to *S*. In this definition *S* is assumed to be the set of design elements related to modeling one view of a system like navigation, composition or visual layout.

The containment meta-model can be formalized further with a rule based point of view. A rule can be used to state that the presence of a certain kind of relation with the containee requires another relation with the container.

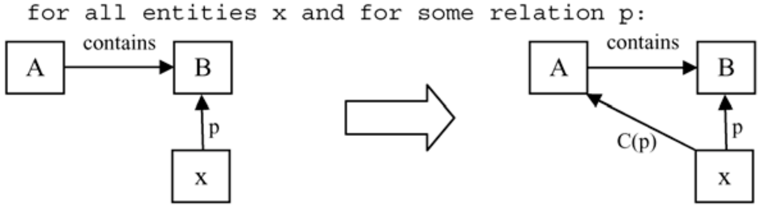


Fig. 4. The rule of containment

Such a rule has been shown diagrammatically in Fig.4. When *node A* contains *node B*, and there is a certain kind of relation called *p* between *B* and a third node *x*, then there must be another relation between *A* and *x*, which is a function of *p*. In the figure, this derived relation is shown as `C(p)`. The characteristics of the function *C*, represents the semantic properties of the containment and each interpretation of containment seems to have its own function *C*. So it is suitable to name *C* as the *containment function*.

Most of the time the *containment function* maps relations onto themselves as `C(p)=p`. Suppose that *A* and *B* are visual components, the containment relation between them is a visual containment relation and *x* is a user. Then `C(visible)=visible` since only when the container is visible the content can be. Now suppose that the component *B* has been created by user *x*. In that case we can say that `C(created)=modified` as by creating a new subcomponent inside it, the component A has been modified.

If the relation with the containee does not cause any relation with the container then the *containment function* should return null. To illustrate this lets suppose that x is an event handler routine that handles the mouse-clicks for B. This relation does not provide any connection between x and A which probably has a mouse-click handler of its own. So C(handles_click)=null gives the value of the containment function for event handling and all the other aspects that are irrelevant to containment similarly yield a null value.

As well as defining and classifying containment at a meta-level, containment modeling should also be able to make statements about the performance of encapsulation in any design instance.

Although we have described encapsulation in Section 2 as an object-wide layering, in fact it is not a real layered architecture since from within the *implementation* it is possible to call any available function in the *outside world*. In the object oriented paradigm the protection supplied by encapsulation works in one direction only. Although as callable procedures the encapsulated methods are being protected by the intervention of interface procedures, as callers they are not.

This policy seems to aim data protection and security and succeeds up to some degree, however it does not totally satisfy the reductionist aims of Taylorism. The interface is a summary of how the object can be used by other objects but it does not summarize how it uses other objects. The compatibility of interfaces does not guarantee that two objects are replaceable by each other and the inner details are not yet encapsulated well enough to be *somebody else's problem*.

Object oriented approach ignores the architectural side of design encapsulation. However this is not affordable in a knowledge dense environment like hypermedia. In hypermedia design one might need to know which and how many links or references from within a container, target the objects that are outside the container. Instead of ignoring the *encapsulation violations* or forbidding them altogether, the reasonable approach is to handle them in the model.

A good way to judge the state of encapsulation of a module is to watch the surface of the module for incoming and outgoing links that penetrate its borders.

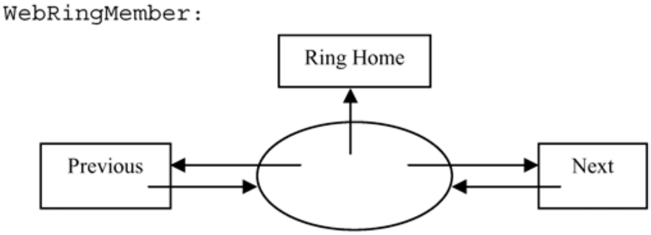


Fig. 5. The controlled encapsulation: Web ring member example

Fig.5. shows a basic design that relies on such a surface control. The design is about the overall navigational structure of a Web site that is a member of a Web ring. A *Web ring* is a collection of Web sites that share a common subject. It is possible to navigate through the ring by moving from one site to another sequentially by links that carry to the next or the previous site. The tour ends when you reach where you have started. There is usually a site reserved as the home of the whole ring and this site can be reached from all the nodes of the ring with a single click. The interesting point about the definition of the design is that it has to rely on controlling the number and type of the links that pass through the boundaries of the site. The definition tells us that a *Web ring member* possesses navigational links to the previous site, to the next site and the home of the ring. However it makes no statement about where inside the member site these links originate from. It also does not introduce any architectural constraint over the *Web ring member*.

Another threat to the encapsulation of a module comes from its own containers when `C(contained_by)=contained_by`. This relation holds for most of the hypermedia artifacts and indicates that the containment relation between them is transitive. In that case a useful validation of an artifact may require looking inside the building blocks of that artifact. As an example one can consider the validation of a type like “illustrated page”. To make sure if the page has an illustration or not, the search should penetrate through nested frames and tables, down to the leaves of the composition tree. Therefore the structure of the containees can be an element of the container's semantics so should not be hidden from it.

In our opinion, regarding containment as a transitive relation by default would be a good choice on the hypermedia domain. However there should also be a way to state that the containment is direct. In fact there are infinitely many possible sequences of intermediate containers between a container and a containee and grammars can be utilized in order to define patterns of such sequences.

4 Conclusions

As the Web keeps entering our lives in a new way each day, software artifacts and programming domains are evolving from modern to postmodern architectures, and hypermedia is becoming a more important design subject. However there is the need that the design tools and concepts go through the same evolution to cope with the high flexibility of hypermedia.

In this work we proposed some enhancements to the concepts of modularity, encapsulation and containment so that the requirements of the new design space can be fulfilled. Throughout the discussion we tried to introduce strategies that can help in the semantic classification of designs in many ways and we tried to emphasize the semantic side of hypermedia by favoring a knowledge preserving approach, instead of reductionism. It is hoped that this work can be a step towards the introduction of new software development paradigms that conform to the postmodern style of production.

References

1. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S.: A Pattern Language. Oxford University Press,New York,1977.
2. Ceri S., Freternali, P. and Bongio, A.: Web Modelling Language(WebML): A Modelling Language for Designing Web Sites. WWW9 Conference, Amsterdam (2000).
3. Conallen, J.: Modeling Web Application Architectures with UML. Communications of the ACM 42(10) (1999) 63-70.
4. Cox, B.: No Silver Bullet Revisted. American Programmer Journal, Yourdon, E.(ed). November 1995.
5. December, J. and Ginsburg, M.: HTML and CGI 3.2 Professional Reference Edition. http://docs.rinet.ru:8080/CGI3.2/
6. Gomez,J., Cachero, C. and Pastor, O.: Conceptual Modeling of Device-Independent Web Applications. IEEE Multimedia April-June (2001) 26-39.
7. Gordon,D. and Whitehead,E.J.,: Containment Modeling of Content Management Systems. Proceedings of Metainformatics Symposium 2002, LNCS 2641, Esbjerg, Denmark, August 7-10, 2002, pp. 76-89.
8. Gu, A.,Henderson-Sellers, B. and Lowe, D.: Web Modelling Languages: The Gap Between Requirements And Current Exemplars. The Eighth Australian WWW Conference (2002).
9. Hall, P., Hovenden, F., Rachel, J. and Robinson, H.: Postmodern Software Development. Open University Computing Department Technical Reports Site, http://mcs.open.ac.uk/mcs-tech-reports/96-07.pdf.
10. Jacobson, M. J. and Archodidou, A.: The Design of Hypermedia Tools for Learning: Fostering Conceptual Change and Transfer of Complex Scientific Knowledge. The Journal of the Learning Sciences, 2000, 9(2), 149-199.
11. Odell, J. J.: Six Different Kinds of Composition. Journal Of Object-Oriented Programming, January 1994, Vol 5, No 8.
12. Schawbe, D., Rossi, G., Barbrosa S.: Abstraction, Composition and Lay-out Definition Mechanisms in OOHDM, in Cruz,I.F., Maks, J. and Wittenburg, K. (eds.) Proceedings of the ACM Workshop on effective Abstractions in Multimedia, San Fransisco, CA (1995).
13. Valdés, M.D., Moure, M.J. and Mandado, E.,: Hypermedia: A Tool for Teaching Complex Technologies. IEEE Transactions on Education, IEEE Education Society, November 1999 Volume42.

The EXTERNAL Experience on System and Enterprise Integration

Weigang Wang[1], Frank Lillehagen[2], Dag Karlsen[2], Svein G. Johnsen[3], John Krogstie[3], Jessica Rubart[4], and Jörg M. Haake[5]

[1] Computation Department, UMIST, Manchester M60 1QD
w.wang@co.umist.ac.uk
[2] Computas AG, P.O.Box 482, N-1327 Lysaker, Norway
{fli,dk}@computas.com
[3] SINTEF, Norway
{Svein.Johnsen,John.Krogstie}@sintef.no
[4] FhG-IPSI, Dolivostr. 15, 64293 Darmstadt, Germany
rubart@ipsi.fhg.de
[5] FernUniversität Hagen, Computer Science VI, 58084 Hagen, Germany
joerg.haake@fernuni-hagen.de

Abstract. The overall objective of the EXTERNAL project is to make it easier for business partners to form and operate extended enterprises as dynamic networked organisations. From a technical perspective, EXTERNAL is largely a system integration project, in which a set of complementary tools are integrated into a Web-based knowledge sharing infrastructure to provide a comprehensive set of services for EE engineering and operation. As a result from our evolving understanding, we shifted from system integration and an Information and Communication Technology (ICT) infrastructure towards enterprise integration and an intelligent Extended Enterprise infrastructure based on Active Knowledge Models.

1 Introduction

Current trends of globalisation and increased competition require new forms of networked organizations. Extended Enterprises (EE) and Virtual Enterprises are two emerging forms of dynamic networked organizations [6,11]. Through shared business goals and work processes, partner-teams manage a behaviour repertoire with associated operational methods, rules and resources to perform business tasks. Engineering an extended enterprise is to model, analyse, and manage extended enterprise models using specific EE meta-models under the guidance of an associated EE engineering methodology. Operating an extended enterprise is to execute the business work processes defined in the EE model and to perform work on the EE models. Important problems in EE engineering and operation are rooted in the poor understanding of how EEs can be developed and operated and how EE development and operation can be supported. This makes EE development and operation currently too slow, complex and expensive. The

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 158–174, 2004.

EXTERNAL project (IST 1999-10091) was a CEC (Commission of the European Communities) funded project aiming to address challenges met when forming and operating an Extended Enterprise [5]. EXTERNAL stands for EXTended Enterprise Resources, Network Architectures and Learning.

The goal of EXTERNAL was to provide repeatable solutions that make the dynamic and time-limited collaboration between EE partners effective. EXTERNAL supports EE engineering and operation by providing:

- A methodology for EE engineering and operation. This methodology includes a description of our active knowledge model approach, visual EE modelling languages and a generic EE engineering process,
- A layered EE infrastructure for distributing and conducting work on active knowledge models,
- Meta-models, models and services supporting the definition, execution, and management of EE models as well as process knowledge management for facilitating organizational learning, and, finally,
- Customer solutions for EE engineering and operation, which were developed and tested in three use cases in the project.

This paper focuses on our work on system integration and our efforts towards enterprise integration as reflected by the development from an ICT infrastructure towards a comprehensive intelligent infrastructure. The rest of the paper is organized as follows: we first describe related work on system integration and enterprise integration. Then, we introduce our active knowledge model approach to EE engineering and operation; and than, we look into more details on our system integration approach. After that we discuss the lessons learnt in the project. The paper ends with a summary and plans for future work. For space reason, details of our EE engineering methodology and the EE visual modelling languages developed in the project are not covered in the paper, but more details are available in our other publications [5,7,8,10,17].

2 Related Work

Businesses and engineering teams today are often faced with the problem of making diverse collections of resources work together. Many organisations have one or more legacy systems that they depend on to operate smoothly. Legacy in this context means not that the systems are old but that they have not been designed with integration to other systems or remote users in mind. The sharing of information on such systems or between different isolated systems may involve exporting data or the re-keying of data [13].

The common methods to integration include custom integration, EAI, component framework, and Web service based approaches [14]. Custom integration involves dedicating current IT resources to the task of achieving some application goal that requires integrated results from multiple systems. Since developers must recode all applications that are impacted by changes, the costs for both maintaining and changing systems can become exorbitant. Most EAI (Enterprise Application Integration) approaches use proprietary integration support systems. It is also very costly for the system integrated in this way to adapt to changes. The traditional standards-based approaches to integration often wrap the system API in some component models for the easy integration in specific component frameworks (such as EJBs, COM, or CORBA, OHS [18]), however the interconnection between systems using different component models and frameworks would lead to complex heavy weight solutions. By enabling standards-based approaches on multiple aspects, such as data format, protocol, registration and searching, the Web Service based integration may be able turn the EAI/B2Bi cost curve down. However, this has to be done correctly. Simply to wrap every legacy or EJBs, COM, or CORBA APIs in SOAP would not help us much. The key to a standard- and service-based integration is to package the systems into loosely coupled services in proper granularity [14,18].

Although we were not able to use Web service frameworks as at the time they were just beginning to emerge, in term of service-based integration, our portal- and worktop-based integration approach is just in the same spirit. It not only uses standard protocol (HTTP) and data format (XML), but also wraps systems into a set of loosely coupled services. Moreover, meta models and tailorable templates and worktops are embedded in the intelligent EE infrastructure to ease the creation of domain-specific enterprise models. The Active Knowledge Model (AKM) visual modelling approach makes the approach intuitive for people to understand, this in turn may make it easier to integrate the EE systems into enterprise partners' daily business. Intelligent Infrastructures, having the ability to vary their capabilities and behaviour from experiences and variations in its environment, are becoming very important. They provide continuously evolving operational platforms for achieving solution interoperability and qualities such as: security, scalability, portability, adaptability, extensibility, and re-use. They support approaches for delivering continuously developing and re-computable IT services and solutions. The EXTERNAL Intelligent Infrastructure is a layered infrastructure, reflecting the layers of Enterprise Architectures recognized by most organizations concerned. So far, little work has been done to implement such Intelligent Infrastructures, and what has been active in research has largely focused on extending the ICT architecture with some meta data in the emerging "Semantic Web" standards and augmenting with distributed system architecture with software agents. But we can benefit from the richer expressiveness and computer process ability of semantic web languages, such as the web ontology language [15] for expressing the EE concepts, relationships, and constraints totally in a machine-understandable format. To our knowledge the EXTERNAL project is the only attempt at implementing an Intelligent Infrastructure using visual Enterprise Modelling to develop and manage the knowledge [3]. Infrastructure qualities will have an immense impact on industrial benefits from ICT technology. Our Intelligent Infrastructure will allow users to benefit from visual scenes of situated knowledge so as to integrate people and systems into an intelligent enterprise.

3 The Active Knowledge Model Approach

The EXTERNAL approach to EE engineering and operation is to create, analyse, execute, and manage live EE models (i.e., an active knowledge model of the EE [5,7]). Fig. 1 depicts the engineering and operation of EEs as dynamic network organizations based on the use of active knowledge models. Various aspects of a dynamic networked organisation (including the organisation structures, information resources, tasks, and processes) are captured as an active knowledge model. Based on the model, business partners in the networked organisation allocate their resources to form a virtual enterprise, in which team members carry out their work by manipulating, enacting and updating the model through visual modelling constructs presented in multiple shared views. While a virtual enterprise ceases to exist after the work is done, the extended enterprise continues to exist. The models represent experiences and can serve as a basis for further joint work.

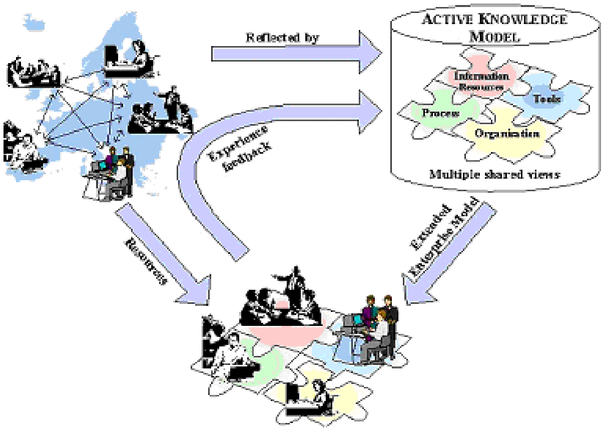


Fig. 1. Extended Enterprise: EE Network and EE Models

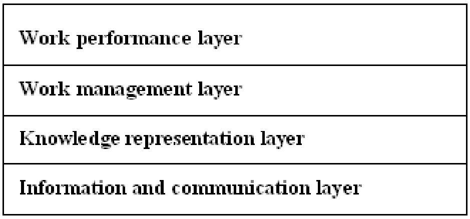
Active knowledge models (AKMs) are built around layered process models, and describe the relevant resources, aspects, views, methods and rules to externalise enterprise knowledge and facilitate knowledge-driven, adaptive collaboration and learning in the EE [7,10]. AKMs are shared, dynamic, co-constructed and extensible, and they are based on open meta-models. Active knowledge models are used for describing all aspects of the enterprise. They are models because they are appropriate descriptions of people's perceptions of organizational reality. They are knowledge models because they reflect collective inter-subjective belief, and finally they are active because they contain and are centred on process models reflecting purposeful organizational behaviour, which are updated and changed as appropriate as work is performed. Thus, active knowledge models describe virtual organizations that can only be activated in an extended enterprise where the participating real enterprises provide the actual resources necessary for knowledge sharing and re-activation [10]. The co-constructed dynamic AKMs of the EE with the allocation of resources from participating partners is used to guide and support the performing extended enterprise in its operation. The dynamic AKM models will, during operation, reflect learning in both the EE and participating partners; thus the growing of and management of a behaviour repertoire will be supported through the management and possible future re-activation of AKM models.

4 EE Infrastructure

For implementing the AKM approach, an EE infrastructure is needed to allow the project to establish open, dynamic work environments for developing and integrating enhanced EE tools, application templates and new operational solutions, and for integrating existing applications into a coherent platform for EE engineering and operation. The ENTERNAL EE infrastructure can be described as a 4-layered architecture, in which each layer offers services to the level above and builds its own service on the services of the layer below [3] (See Fig. 2). These layers are described as follows (in the bottom up order):

- The Information and Communication Technology (ICT) layer is composed of basic IT architectures, tools and software components, and applications. It ensures inter-operability, scalability, connectivity and communications. The ICT layer is there to support multi-user access, concurrency control, access control, versioning and configuration management, etc. It is a 3-tier architecture, the tiers being the clients, the application servers, and the data servers.
- The Knowledge Representation layer defines and describes how model, template-objects and meta-data are represented, used and managed. It provides a set of generic AKMs to be used as solution templates. It supports a meta-data hierarchy for meta data management. An EXTERNAL Common Type (ECT) DTD has been defined and is supported by each of the enterprise tools integrated into the EE infrastructure. The models are persistently stored in a shared model repository. The models are created using the EE modelling language developed in the project. This language includes domain specific object types, which can be derived and extended from basic ECT types.
- The Work Management layer provides work processes for modelling and model management. It provides services and work processes for change and life cycle management. It also provides services for cooperative work in a team- and role-based cooperative working environment.
- The Work performance layer provides a worktop-based user environment. This is the high-level application layer directly exposed to end-users [1,7,8]. The worktops can be configured based on the contextual situations and personal preferences as defined in specific model templates and interface layout templates. These worktops set up the right context for a certain task and identify the resources needed to do the task. The work processes are made available and executed in the background through these worktops.

The Knowledge Representation layer, through meta-models, provides the binding glue between all the system components. It defines the interfaces and data exchange formats between the components, and specifies how changes of the models in the repository feed back into changes of the worktops (more specifically, the templates they use, the information resources they display, and the tools they offer on these items).



Work performance layer

Work management layer

Knowledge representation layer

Information and communication layer

Fig. 2. Intelligent EE infrastructure

5 Original EE Tools to Be Integrated

The EE tools integrated into the EXTERNAL infrastructure at this time include:

- METIS [7] a general-purpose visual modelling tool that helps people use complex enterprise knowledge to answer critical questions and solves business problems,
- XCHIPS [17], a cooperative hypermedia tool providing shared hypermedia workspaces integrated with process support and support for synchronous collaboration,
- Vité/SimVision [12], a project simulator used to analyse resource allocation, highlighting potential sources of delay and backlogs,
- WORKWARE [9], an emergent workflow management system with Web browser-based work list handling, document sharing, and process enactment mechanisms,
- MS Office tools for information processing and management, and
- MS NetMeeting and other groupware tools (in the XCHIPS toolset) for communication and other task-specific cooperative activities.

Both METIS and XCHIPS have certain meta-modelling capabilities, which support the adaptation of EE modelling schemas, templates and other meta data. METIS and Vité/SimVision are single user tools. XCHIPS is primarily designed to support synchronous cooperation. It can also support a combination of or smooth transition between synchronous and asynchronous cooperation as the shared data is persistently stored in a database on its server side and changes made to shared data are sent to clients, which are interested in this data. Thus, it is very useful for negotiation tasks. WORKWARE provides a thin Web client and supports asynchronous cooperation through a shared repository at its server side focusing on process execution.

6 System Integration and Development

Connecting and integrating systems of the business partners in an extended enterprise is of critical importance to reduce business costs, to improve customer service, and to facilitate closer business relationships. However, integration remains complex, expensive, and risky. Typically, integration of applications involves communicating between various resources (such as components, applications, devices, data sources, data formats, and protocols), coordinating their operation, and extending their basic capabilities with additional functions.

One of the technical scenarios that can benefit from the integration of the above-mentioned complementary tools is the following: METIS is used for creating visual enterprise models, and XCHIPS is used on selected parts of the model in a virtual meeting setting for cooperative inspection and modification. SimVision can be used from time to time to analyse the model and to identify areas for improvement (i.e. to call on XCHIPS to modify the model and then simulate again until some degree of satisfaction is reached), and finally deliver the model to WORKWARE for enactment. During the execution, SimVision may still be used for simulation and analysis and XCHIPS may still be called upon for modifying the underlying EE model. Office tools will be activated by document processing related tasks. NetMeeting may be invoked for application sharing and A/V communication. Other groupware tools, such as the ones for brainstorming and for architecture construction may be activated by cooperative tasks that carry out such activities.

The following sections describe how the above four enterprise tools, standard office tools and web browsers, as well as groupware tools, are integrated into the EXTERNAL infrastructure

6.1 Integration of Enterprise Tools

For enterprise modelling, simulation, and execution tools, the integration proceeds in three steps [3,16]:

- Data-centred integration: based on a common EXTERNAL XML DTD, XML importing/exporting utilities are implemented in each of the enterprise tools for data exchange between the tools or between an XML repository and the tools,
- Control-centred integration: this is done by using the APIs provided by the tools and the repository to be integrated. With the APIs, the tools can call each other and access the shared repository. Some of the APIs may have parameters for denoting content objects and the implementation of them requires the data-centred integration capability as developed in step one, and
- Worktop-based integration: this is a service-based integration that makes use of both data-centred integration and control-centred integration methods to access shared models, information objects, and to invoke individual tools. The worktops can be configured based on the context of the current

situation and on personal preferences as defined in specific model templates and interface layout templates.

The following sections provide more details on the integration effort.

Data-Centred Integration: With a common data format and a shared model repository supporting this format, different tools can be used together or in any sequence to manipulate information or documents in this format. The common EE modelling language XML DTD developed in the project (see section 6.5) has built a solid foundation to integrate our existing tools and to access EE models in the shared XML repository (see Fig. 3. The arrows in Fig. 3 indicate data flow between tools and the shared repository).

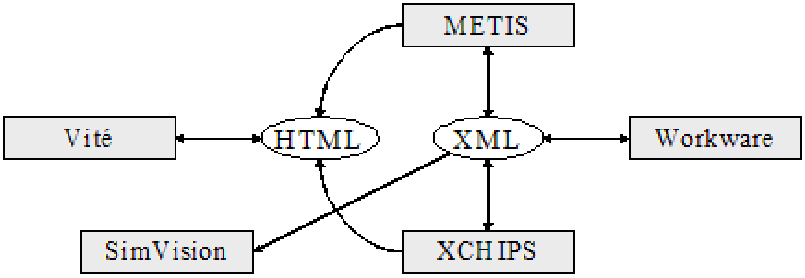


Fig. 3. Data-Centred Integration

Based on this XML DTD and the mapping methods described in section 6.5. XML importing and exporting utilities for METIS, XCHIPS, and WORKWARE have been developed. These XML import/export utilities have been tested in a demo use case and were refined based on feedback and changes made in the EXTERNAL DTD. As the earlier version of Vité (i.e., VitéProject2.2) uses Microsoft ACCESS as its back-end, we developed an HTML exporting utility from XCHIPS, and these HTML files were linked into the ACCESS database for VitéProject to access. Later, when VitéProject3.0 (SimVision) became available, we developed a mapping tool to translate models in the common DTD to the XML format used by SimVision. This mapping tool uses an XML style sheet to describe the mapping between the elements in the common DTD and the elements in the SimVision XML DTD.

Control-Centred Integration: Using open APIs published by collaborating tools and the shared repository, tools can access the repository and call each other directly. Specifications for invoking and controlling applications through open APIs have been drafted by each tool partner and distributed to each other. With these, users can invoke Vité/SimVision for simulation, and call METIS for modelling and WORKWARE for task execution. Users can also call XCHIPS on specific objects from METIS and WORKWARE. HTTP-based APIs of XCHIPS, METIS, and WORKWARE have been developed and implemented. Based on these APIs, a coordination worktop has been developed in the Web portal of the EXTERNAL environment, from which all the tools and their selected service parts can be activated in their application context (see Fig. 4).

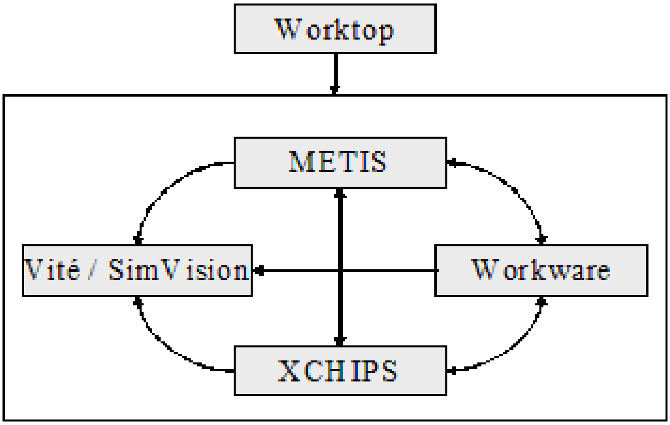


Fig. 4. Control-Centred Integration

Worktop and Service-Centred Integration: For wide availability, the EXTERNAL infrastructure uses portal technology to build its user environment. The environment is highly customisable and has capabilities to integrate tools as services in personal or project specific work areas. The interfaces from the portal to the main services are solely based on the HTTP protocol, which is also the case between the main services themselves.

The EE Portal Server runs on a Windows 2000 Server using IIS as web server, and is based on ASP scripting and relay to an SQL Server 2000 database. The worktop environment connects standalone tools using both data-centred and API-centred approaches (see Fig. 5). Before being integrated into the portal and worktop environment, each tool is repackaged into one or more concrete services, each of which provides an HTTP-based API. In this way, we avoided to make the tools unduly tightly coupled with too many APIs and provided a set of loosely coupled services of proper granularity. For instance, the XCHIPS system is divided into a cooperative model editing service, a hierarchical view service, a task control panel, and a set of general groupware services (such as white boarding service, brainstorming service, chat service, etc); and the METIS tool is divided into Visual modeller, Model viewer, Model annotator services. The user interface of the worktops can be configured based on the context of the current situation and on personal preferences as defined in specific model templates and interface layout templates.

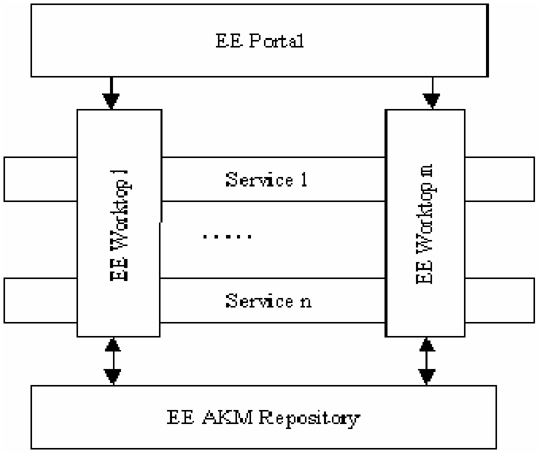


Fig. 5. Integrated Worktop Environment

In Fig. 5, the EE Portal is a common point of access to the enterprise modelling and working environment. It is from here, where all users start working in their EE. The services provided include services supporting generic tasks such as modelling, simulation, work performance, text editing, document handling, and application-sharing activities. Since the actual resources and services needed for performing such activities heavily depend on the current state of operation and of the individual user, worktops must be dynamically configured at run-time. By working in a worktop, users will create and manipulate information resources in the EE and thereby change the state of operation of the EE, thus potentially leading to further change of (future) worktops of other users.

6.2 Integration of Standard Office Tools and Web Browsers

Microsoft office tools are integrated into the EXTERNAL infrastructure through their COM interfaces. Object wrappers were developed for each of the office tools to be integrated (for instance, one for Word and one for Internet Explorer). For tools without a COM interface, the navigation method of the Word wrapper or run-time application invocation functions is used. For tools with a standard Web browser front-end, helper applications can be specified for different file types.

6.3 Integration of Groupware Tools

In the XCHIPS system, a groupware component wrapper was developed. This wrapper can be created at any place in the XCHIPS shared information space. There is a list of groupware components pre-loaded to the XCHIPS server and available for all clients. Users can also add new groupware components dynamically at run time for all clients to use them. Groupware components (i.e., groupware tools, such as a whiteboard, chat tools, and shared editors) can be plugged into the groupware component wrapper with a copy-and-paste operation. Users can invoke a groupware tool by executing the “open” operation of the wrapper.

For voice communication and for application sharing, MS NetMeeting has been integrated in XCHIPS using a groupware component wrapper. This groupware component can be launched from the XCHIPS toolbar for people to set up a NetMeeting conference automatically (as XCHIPS’s “session” object knows its users, and “user” has an attribute for the IP address of the user’s client. The value of the IP address attribute is automatically set in the user login process).

6.4 Shared Model Repository

In the project we implemented two alternative repository solutions: 1) The METIS Team Server as a shared model repository and 2) a WebDAV-based repository. The first is a solution that is highly scalable both in number of users and performance, while the second is meant for smaller installations. The shared model repository is used to store models and documents for all projects. The models are logically stored as XML-files based on the DTD developed in the EXTERNAL project.

6.5 Further Details on the Data-Centred Integration

The Common Object Model: Originally, the four initial EXTERNAL tools were developed independently by different partners. Although there is a great overlap in their representational domain , naturally each has their own representations of the various aspects of this domain.

The Common Object Model (COMod) [3] is a tool-independent representation of the objects, types and their interrelations, applicable throughout the Extended Enterprise infrastructure and tools and may be thought of as a representation of the shared part of the Extended Enterprise Modelling Language (Common EEML) [8]. The Common Object Model is developed using a mixed bottom-up/top-down approach. Mappings between the Common Object Model (Common EEML) and the modelling languages of the individual tools are documented. Based on COMod, an XML-DTD is defined, which represents the common storage format for data in EXTERNAL. The XML-DTD is semantically more limited than the COMod. Therefore, to establish a correct mapping from the XML-DTD to the modelling language of each of the tools one has to take into account the constraints of the COMod that are not expressible in the XML-DTD. The stippled lines in Fig. 6 below illustrate this.

The EEML is thoroughly described in the EE Methodology deliverables (D2/D6) [8]. Basically, it covers quite a few aspects that one would want to model of an extended enterprise. However, only the process oriented concepts and relationships may be used throughout the EE infrastructure, i.e. shared between tools. In Fig. 6, this subset of EEML is called Common EEML.

The Common EEML is semi-formally specified as a UML model, using the following UML constructs:

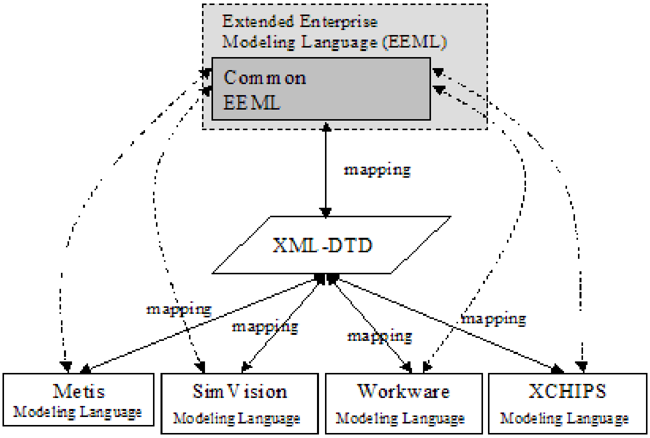


Fig. 6. various representations of process models, and the mappings between them

- EEML concepts, both abstract terms and language terms of both object and relationship types are represented as classes;
- Taxonomic relationships between all concepts are represented by the specialisation relation;
- Any other relationships between concepts are represented by associations;
- Domain membership is expressed by the class stereotype; and
- Constraints on relationships are represented as constraints. In addition to the UML model, there exists a natural language description of the language terms of the Common EEML, including things omitted in the UML model, e.g. further specification of constraints and graphical symbols.

As the above indicates, the Common EEML model does not exist in any machine-readable form, but as a systematic specification, which serves as a guide during development of the mappings between the common storage format (XML DTD) and the individual tools.

Developing the XML DTD: As a widely used standard, XML was chosen as the format for the data integration and exchange in the EE infrastructure. In the specification and implementation of the Shared Model Repository (SMR), developing a DTD became the core process.

The current SMR implements a subset of the EEML, represented as the Common Object Model (COMod). The process of developing the DTD consists of, for each model element in COMod, the identification of suitable XML constructs corresponding to each element. In addition, some implementation constructs were added. Extensibility is an important aspect of the EEML, and had great influence in the main principles followed in developing the DTD (often referred as the EXTERNAL Common Type Definition - ECT DTD). These principles are:

- Broadness of scope: The DTD should implement all elements in COMod;
- Stability: The DTD should be specified in a manner that ensures stability of the DTD; and
- Extensibility: The DTD should be specified in a manner that ensures extensibility in the EEML.

The limited expressiveness in XML implies the need for additional means for expressing the implementation of the COMod. Constraints that cannot be represented by a DTD are documented in UML and natural language descriptions, and the applications using the DTD need to take care of validation themselves, whether it is data type checking, value range checking, checking of relationship constraints or other possible rules.

In the DTD all object types are implemented as XML elements. The core object types (i.e. tasks, roles and other related object types) are implemented as specific XML elements. In addition, two generic XML elements are declared to ensure the extensibility at the object and relationship type level, respectively.

Attributes are partly implemented as specific XML attributes and sub-elements, and partly by generic sub-elements declared to ensure extensibility at the attribute level. Each of the generic sub-elements has attribute pairs, one for the attribute name, and one for the attribute value.

Some of the relationship types have also been implemented as specific XML elements, while others have been implemented as sub-elements that contain URI's to other XML elements. All decompositions have been implemented as sub-elements with URI's as values.

All relationships that are implemented as specific XML elements have - in addition to the id - a sub-element called origin and a sub-element called target. They point to the element implementing the start object and the end object of a relationship, respectively.

The development of a detailed description of the mapping between the COMod and the DTD was important in the process of implementing the COMod in the DTD. The mapping is described in the next section with details. As the EEML is extended, our approach in implementing the COMod in the DTD ensures stability of the DTD. In most cases, added model elements are reflected in changes in the mapping between the COMod and the DTD while leaving the DTD unchanged.

The Mapping between the COMod and the DTD. The mapping between the COMod and the DTD constitutes the link between the COMod and its implementation, and is the core documentation of the implementation in the DTD. The mapping description is divided into the following parts:

- Overview of the mapping on object and relationship type level. This is presented in a tabular way, with one column for the object and relationship type in COMod and one column for the corresponding element type in the DTD. Entries in the same row represent one mapping. A third column contains additional specification when needed.

- The mapping of the object types at the attribute level. The presentation is done in a similar way as the mapping overview. The attributes are listed in groups by the owner object type.
- The mapping of relationship types at the attribute level. The presentation is done in a similar way as the mapping overview. The attributes are listed in groups by the owner object type.

Implementing the Common Language Model in Individual Tools. The development of the EXTERNAL enterprise modeling language, COMod, the DTD, and their mapping to the modeling languages of the individual enterprise tools is an iterative process. We first studied the modeling languages of our existing tools (i.e., METIS, XCHIPS, Vité/SimVision and WORKWARE) and other enterprise systems. Then, by extracting the common aspects from the languages, we developed versions of the EXTERNAL common enterprise modeling language, which as described above is documented in UML, natural languages, and a graphical language of our own. For storage, data-exchange, and interoperation between systems, we developed an XML DTD representing the syntax aspects of our common modeling language (COMod). Finally, mappings between the common modeling language and the DTD to individual modeling languages of individual tools (or systems) are developed.

More specifically, on individual system side, a (partial) mapping between the common DTD and the DTD (or types if there is no DTD) of the individual system is defined. Then, translators (or importing/exporting utilities) for the mapping between the ECT DTD and the DTD (or types) of individual systems are developed. For the systems that already have a DTD, the translators (i.e. the mappings) are developed using XML style sheets. For the systems that do not have a DTD before the integration, the translators are developed using an intermediatory object representation of the (partial) mapping(s) in Dynamic XML (DXML). Based on the DTD, each tool can define an import utility and an export utility once for all and each tool that understands the DTD can access its corresponding XML repository for manipulating the shared enterprise models.

7 Lessons Learnt

The AKM based Visual EE modelling languages developed in the EXTERNAL project are semantic rich. They offer a large amount of (extensible) object and relation types on various aspects of an enterprise and have structural and relational constraints and operational semantics. Due to its limited expressiveness, the XML DTD developed in the project can not represent all the needed information on data types, constraints on relationship, and the mapping between symposiums of concepts in the DTD and the individual systems. In EXTERNAL we used several additional representations, e.g., UML, natural language descriptions, and mapping tables. Many of these representations are not computer processible, and therefore, the knowledge represented in such forms has to be interpreted by the system developers and hard-coded into the individual systems to be integrated. These hard-coded parts made the system difficult to couple with changes on that part of the knowledge. We need to find better knowledge presentations that are computer processible to solve the problem.

Another lesson we learnt on the EXTERNAL DTD is that we might be better off if we represent all the objects and relationships using the general object and relationship elements, rather than as we did now with some of them represented using specific elements and the rest using the general elements. Although the specific elements made their semantics more explicit for the users, they are inflexible for change management. Because when these part of object types changes, the DTD has to be modified.

On the positive side, the use cases of the EXTERNAL project demonstrated the applicability of our general approach to system integration and enterprise integration, and highlighted the usefulness of our EE Intelligent Infrastructure. For better system integration, we have to understand the models and meta models of each enterprise system to be integrated, so as to identify the common aspects (i.e. an integrated model: the kind of meta of meta). Further more, to support a higher level integration – enterprise integration, we have to embed enough enterprise knowledge into an intelligent infrastructure, so as to make it easier for enterprises (and their systems) to join or extend themselves into an extended enterprise. This high-level knowledge of enterprise belongs also to the meta of meta grade, which reflects the common understanding or shared knowledge of the enterprises to be integrated.

8 Conclusions and Future Work

Taking an active knowledge model approach, EXTERNAL aims to support process collaboration with embedded knowledge and change management: creating, sharing, harvesting, and re-activating knowledge. This requires development of methods, infrastructure/tools and business solutions for Extended Enterprise modelling, analysis, engineering and operation. The project thus targets the integration and convergence of currently fragmented business process technologies. Using EXTERNAL results, companies will be able to transform, extend and tune their operations. The innovative contribution of the project is reflected in the following aspects:

- Dynamic networked EEs are integrated by shared active knowledge models;
- The Methodology and the Infrastructure support emergent processes with integrated visual modelling, co-operative planning, scheduling, and performance of work; and
- The EXTERNAL approach facilitates reuse of sub-models and model fragments, and domain constructs to compose new models, and to support dry runs to simulate and validate proposed patterns of co-operative actions.

Learning in dynamic networked organizations is supported as harvesting of lessons learned from process histories, in order to build repositories of reusable process model templates. Employees can learn about current and planned processes and how to perform them by using the shared EE active knowledge models.

In the EXTERNAL project, a large amount of engineering work has been devoted to integrate our existing enterprise modelling and operation tools into a four-layered intelligent infrastructure. In order to cope with the changes in the dynamic business environment, we have tried to go beyond a simple custom integration by uncovering meta models of each tools and extracting the common aspects into a shared meta model. This shared meta model is then used as the basis not only for integrating the existing and newly developed systems, but also for establishing shared understanding among all the business partners in the project consortium that we run as an extended enterprise. Such shared understanding is further enhanced by the joint creation of domain specific enterprise models using the intelligent infrastructure. Our experience tells us that shared understanding is the basis to integrate people from different business partners into a united whole and such integration is the key for enterprise integration. As a result, an agile and intelligent EE can be formed and operated [13].

The EXTERNAL infrastructure has been used at all the use case partner sites for the three use cases carried out in the project. Such use cases have informed the development and refinement of the methodology and the infrastructure. The work on the EXTERNAL Intelligent Infrastructure continues both in commercial projects and in RTD projects. Some of the existing prototypes are already being re-implemented as commercial capabilities to be delivered with the METIS suite of visual enterprise modelling, architecting, and integration products. One line of work we are investigating is to repackage our EE services using Web services. We are also very interested to explore how to benefit from the Semantic Web initiative [2,4], especially for the richer expressiveness and computer process ability that we need in supporting our AKM knowledge model driven approach to system and enterprise integration. In this way we may be able to further improve our approach and implementation so as to couple with changes better.

Our longer-term focus will be on product and process design, multi-project management, and on new ways of performing business. We will also focus on re-implementing what is today legacy systems and on developing more tailored solutions to many of the industrial challenges and promises that the IT industry has made since the mid 80'ies. Research on model-driven computing services to replace today's manual operating services is another line of work we want to follow.

Acknowledgments

We thank all the team members of the EXTERNAL project for their contributions to the project.

References

1. Afsarmanesh, H., Garita, C., Hertzberger, L.O., and Santos-Silva, V., Management of distributed information in virtual enterprises - the prodnet approach, 4th International Conference on Concurrent Enterprising, 1997.

2. Berners-Lee, T., Hendler, J., and Lassila, O. (2001) The Semantic Web. Scientific American, May http://www.sciam.com/2001/0501issue/0501berners-lee.html
3. Karlsen, D., et. al, D7 EXTERNAL Infrastructure available at www.EXTERNAL-IST.org
4. Decker, S., Melnik, S., Harmelen, F. V., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., and Horrocks, I. (2000) The Semantic Web: The roles of XML and RDF. IEEE Internet Computing, Vol. 15, No. 3, October, 63-74
5. EXTERNAL consortiums, The EXTERNAL project, March 30, 2003, www.EXTERNAL-IST.org
6. Larsen, Kass-Pedersen and Vesterager, Creating a generic extended enterprise management model using GERAM - Presentation of the IMS/Globeman21 project, IsoCE 1998 conference, Sinaia, Romania, June 1998.
7. Lillehagen, F., Karlsen, D., Visual extended enterprise engineering and operation-embedding knowledge management and work execution, Production Planning and Control, 2001, Vol. 12, pp. 164-175
8. Krogstie, J., et. al, EXTERNAL methodology reports D3 and D6 available at www.EXTERNAL-IST.org
9. Jørgensen, H. D. and Carlsen, S. Emergent Workflow: Integrated Planning and Performance of Process Instances, Workflow Management'99, Münster, Germany, 1999.
10. Krogstie J., Jørgensen H., and Lillehagen F.: Active Models for digitally enabled creative business networks, Paper presented at IFIP World Computing Conference, Montreal, Canada, 2002.
11. Krogstie, J. and Carlsen, S. An Integrated Modelling Approach for Process Support, Hawaii International Conference on System Sciences (HICSS-30), Maui, Hawaii, 1997.
12. Kuntz, J. C., Christiansen, T. R., Cohen, G. P., Jin, Y. and Levitt, R. E. The Virtual Design Team: A Computational Simulation Model of Project Organizations, Communications of the ACM, vol. 41, no. 11, 1998.
13. Hailstone, R., et. al, IT and Business Integration: Moving towards the Agile Organization, Feb., 2003, An IDC White Paper Sponsored by Exact Software
14. Schmelzer, R., Services and XML to Integrate Systems, ZTR-WS103, October 23, 2002 17. Ronald Schmelzer, Understanding the Real Costs of Integration, 2003, at www.zapthink.com
15. Van Harmelen, F., Patel-Schneider, P. F., and Horrocks, I. (eds) (2001) Reference description of the DAML+OIL ontology markup language, March http://www.daml.org/2001/03/reference.html
16. Wang, W., et. al, D8 CD documenting and installing the Infrastructure and partner tools, available at www.EXTERNAL-IST.org
17. Wang, W., Haake, J.M., Rubart, J., Tietze D.A. (2000) Hypermedia-based Support for Cooperative Learning of Process Knowledge. In: Journal of Network and Computer Applications, Vol. 23, pp. 357-379, Academic Press
18. Wiil, U.K., Hicks, D.L., and Nürnberg, P.J. Multiple Open Services: A New Approach to Service Provision in Open Hypermedia Systems. In Proceedings of Hypertext'01, ACM Press, 2001, 83-92.

Interaction with Information Technology Seen as Communication

Chickens and Eggs and Meta-applications

Frank Wagner

Roskilde University, Universitetsvej 1, 4000 Roskilde, Denmark
frankw@ruc.dk

Abstract. Applications are still getting more powerful and complex. This is potentially useful for the user, but it puts a burden on the users too. Information handled by one application can be difficult to use with a different application. Developments in structural computing could be useful to address this problem, but applications that use structure services and make them available to users are rare.
Most of the things I am going to say are or should be rather obvious. The intention with this paper is to try a slightly different view and to emphasize an interaction between IT and users that has been neglected.

1 Chickens and Eggs and Meta-applications

In this paper I comment on two points from MIS'02 [1]. The first one is about the discussion about the roles of structure and data. As with the discussion about what came first, the chicken or the egg, it might help to step back to get a better view. Instead of trying to find rather specific definitions of these terms, I want to look at information and how we use it. The other point is about applications and the roles of both their users and their developers. Applications are build by the developer to handle certain kinds of information. How does an application look like, that helps users to works with information in general (including information about the kind of information) and who can develop it?

The idea is to find some kind of basic concepts, that can be common to both IT and us as potential users of IT. The implementation of these concepts will be quite different because of the different kinds of system, but common concepts should help, especially when users are going to be more active in the development.

2 Information and Behaviour

Behaviour is another term that has been discussed at the MIS'. In this section I want to mention a few aspects about both information and behaviour that I consider to be relevant for applications. Maybe the most important point is that information and behaviour are inseparable. They determine, modify and supplement each other. This is in contrast to the relation of functionality and values.

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 175–177, 2004.

Information is about something. Information stresses certain aspects of some object. This object can be a physical thing, a thought, another information or whatever. We use the information to deal with the object. The particular form or representation of an information depends on where it comes from and what it is to be used for, but it is not relevant in this context.

Behaviour is how we react in a given situation. We use our senses to get information about the situation. We interpret this information and use the interpretation to react. We can express and represent information, so we can exchange, safe and reuse it. Structuring information leads to other information and related behaviour.

An important aspect of both behaviour and information is when and how we can access and use them. We can not be aware of everything at the same time, instead, in a given situation we choose a certain view and focus on information we consider relevant. Information out of focus still is relevant though, because it has formed our behaviour. Being aware is part of our behaviour.

3 Applications

IT is used to help carry out different kinds of tasks. These tasks involve manipulation of information. Both the tasks supported and the information handled have been becoming more and more complex over time.

Computation of values does not require any understanding or interpretation of these values. It does not matter what the values are used for, only the rules for treating the values are of interest. Examples are simple spreadsheet or textprocessing applications or a calculator application. Such applications provide some functionality.

Working with data requires an understanding of how the data is related to each other. The developer of an application analyses the task that is to be supported and builds a data model. The application has to ensure dependencies and constraints. A simple example can be an addressbook application. This kind of applications still provides functionality.

The kinds of applications mentioned both put a load on the user, especially when a user has to use different applications at the same time. The user is responsible for organising different information and the user has to prepare the information so they can be manipulated with the different applications. He could use a meta-application to deal with the other applications. Applications that work on information in general could take some of this load off the user.

4 Interacting with Information Technology

The application that I am missing is an application that helps to deal with generic information. It has to deal with information about information. Information will be supplied and used by the user and it should be supplemented by the application. The degree to which the application can contribute depends on the extent to which it can attach functionality to the information, that is how the application can behave. A very important part of the applications behaviour is to identify information and to trigger the behaviours related to this information. Triggering a behaviour means to find some basic functionality and to modify it according to the given metainformation (information about the information and about the context). This basic functionality is provided by services. Identifying information involves finding structures. Several different structures can be useful. A spatial structure service can provide the functionality needed by behaviours that analyse and build the interface. Taxanomical and ontological services can support behaviour finding context information. A navigational service can help to maintain trails and provide links to related information. An important point is, that the services only provide the functionality. The actual behaviour is determined by additional information about and from the information, the context, the application and not at least the user.

A problem with many of the existing end user applications is, that they interact with the user directly and that they only deal with information internally. The application is between the user and the information. The information is availble only in an application specific representation. Other applications can access this information, but the connection to the user or more general the origin is lost. Possibly relevant behaviour can not be attached. Instead of using their own interface, they should reuse existing information and the related behaviour. An even better solution would be to contribute functionality as a service. The application can and will still have its own internal representation though. The information as it was perceived through the interface could make the process more comprehensible. Logging and versioning could be managed in a consistent way.

5 Conclusion

Focusing on information instead of tasks leads to a different description of the levels of work discussed at the MIS'02 . A-level work deals with the application and preparation of information. B-level work deals with the characteristics of information and how information can be providet. C-level work deals with the underlying funtionality. Even though A-level and B-level work deal with information, the the focus and the point of view are different.

Considering the amount of available information, IT is going to play an active role. The trend to personalisation demonstrates this [2]. The question is how we can control this. Regarding the interaction with IT as communication can be useful. Communication with technology will be missing many of the qualities that we know from communication between people, but then again we do not have to expect to achieve the same results either.

References

1. Nürnberg, Peter J. (Ed.) Metainformatics Symposium 2002, August 7-10, Esbjerg, Denmark *Proceedings of Metainformatics Symposium 2002, Lecture Notes in Computer Science, Vol. 2641*, Springer-Verlag, Berlin Heidelberg, August 2002.
2. Tolksdorf, Robert und Eckstein, Rainer (Herausgeber) Berliner XML Tage 2003, 13.-15. Oktober, Berlin, BR Deutschland *Tagungsband Berliner XML Tage 2003*, Freie Universität Berlin, Humboldt-Universität zu Berlin, Oktober 2003.

Meta-analysis and Reflection as System Development Strategies

Christopher Landauer and Kirstie L. Bellman

Aerospace Integration Science Center
The Aerospace Corporation, Mail Stop M6/214
P.O. Box 92957, Los Angeles, California 90009-2957, USA
{cal,bellman}@aero.org

Abstract. This paper is about strategies for developing new approaches to solving difficult problems in complex computing system design, based on a very strong use of abstraction and reflection. We address problems in the design and development of Constructed Complex Systems, which are large, heterogeneous, usually distributed, systems managed or mediated by computing systems. We call this strategy "meta-analysis", since it is as much about the processes of modeling and defining elements of these systems as it is about new kinds of elements themselves.

For some years, we have had interesting successes in defining new approaches to these problems, and we have recently noticed that there is some commonality in the approaches.

We define what meta-analysis is, show some examples from our earlier research, and explain why it is a meta-informatic approach.

1 Introduction

This paper is about *Constructed Complex Systems*, which are heterogeneous distributed systems managed or mediated by computing systems. We expect that any system that is designed to operate in a complex environment, or that is intended to interact with unforeseen situations, will have to be a Constructed Complex System.

We describe our strategies for developing such systems, and particularly how our approach is based on models. We explain how the model-building strategies are based on the meta-informatic principles of "meta-analysis" and "reflection", and why we think those are important. While the strategies have been in development for some years, some of their commonalities, and their relationship to meta-informatics, was not heretofore recognized.

Most system development methodologies assume that the domain is well-understood, so that there can be a consistent set of definitions of important domain terms and concepts, operations and relationships. The best of these approaches relies heavily on appropriate mathematical formulation [12] [22] [28], but other more systematic [11] [14] [27] or computational [16] (and less mathematically formal) system development methodologies also emphasize the care

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 178–196, 2004.

with which models are to be defined and constructed, but none of them sufficiently emphasize the analysis and evaluation of models. In addition, all of them suffer from the problem of making too many choices too soon [58].

We are mainly interested in the application areas for which this domain assumption is not true and may never be true, so that all of the fundamental definitional elements will change, repeatedly, before, during and after system design and construction, delivery and operation. It is our opinion that in these situations, the system will have to do much of the modeling of its own structure, behavior, and effectiveness, and it is our observation that these properties require much more explicity and autonomy on the part of the system, and much more abstraction on the part of the system design.

We take a strongly mathematical approach that generally requires developing or applying new mathematics. Our most interesting sources at present are Category Theory [48] [63] [20] [44], especially applications of Co-Induction [23], and Non-well-founded Set Theory [2] [5], both of which are very recent (as mathematical theories go).

We argue that the key to understanding and managing any complex system is the models of it that we can examine. These "model-integrated" [45] [46] and "model-based" [31] [42] systems make essential use of models in their behavior. For Constructed Complex Systems, we also expect the system to be able to examine at least some of the models.

In our view, active models are best considered as passive declarative models with active interpreters, and we claim that maintaining the separation between them is important. Of course, declarative knowledge does nothing by itself; it requires active processes that interpret the knowledge to provide the "spark of agency", accounting for all activity in a system. This property is why the processes are as important as the knowledge bases in our Wrapping approach described below. We only need the descriptions to be sufficiently complete, that is, to describe all of what the processes do, in some modeling notation or notations that can be interpreted by other processes (which must of course also be part of the system, in order to close the loop).

We start by describing our terms "meta-analysis" and "reflection" in the rest of this Section. In the rest of the paper, we describe some of our research in Constructed Complex Systems, and show how it has benefited from our meta-analytic approach. Their importance for this paper is not in the specific topics; it is in the common meta-informatic principles that we apply in our studies, that have led to success in each case.

1.1 Meta-analysis

We start with a definition of what meta-analysis means to us: it is a generic term for a kind of model definition process that includes some of the main modeling steps [57] [8]

- collecting and describing phenomenology
- selecting or constructing of formal space
- defining mapping to and from formal space

- (the other modeling steps are
    - performing manipulations in the formal space
    - mapping resulting relationships and activities back into the application domain)

and also includes some other processes

- abstraction
- setting boundaries
- noticing interactions
- naming elements
- defining notation

The most important aspects of this modeling process are in the first step: to identify key structures and behaviors, and to identify key relationships and interactions.

For structure: the usual decomposition of structures in a complex phenomenon is a kind of principled decomposition of those structures into substructures with defined relationships. This is not the only kind of decomposition, however. In addition, decomposition can often reflect historical relationships that are not easily defined according to apparent principles.

For behavior: the usual decomposition of behaviors in a complex phenomenon is a kind of principled decomposition of those behaviors into subsidiary behaviors with defined interactions. As above, this is not the only kind of decomposition, however. In addition, decomposition can often reflect historical interactions that are not easily defined according to apparent principles.

Because it is not possible, even *in principle*, to get all of these decompositions right the first time, it is essential in our systems that change be possible, and that it be possible in a way that the system itself can support. This expectation leads to our interest in Computational Reflection.

1.2 Computational Reflection

Computational Reflection is the ability of a system to have access to its own structure and behavior [61] [49] [50]. Most approaches to these systems choose a subset of system activities that are expected to require change, and design the “meta-object protocols” [26] that allow them to be changed under control of the system user(s) (e.g., “this use of the algorithm is much more effective with implementation X than with implementation Y”). Systems with Computational Reflection can reason about (some of) their own processes, and make activity selections based on their conclusions (e.g., “if I knew fact X, then I could perform process Y more quickly when it is requested of me, so I should go find fact X when I have time”). Some systems with Computational Reflection can even reason about (some of) their own reasoning processes, and make activity selections based on their conclusions (e.g., “I do not have enough information at hand to conclude fact X, so if I have asserted fact X, it is either externally provided by a reliable source, or it is hypothesized, and should collect as much corroborating evidence as I can”).

Our approach to Computational Reflection is based on our intended application area of autonomous Constructed Complex Systems in complex environments [38]: our systems are all model-based [31], and we expect the models to carry all of the essential information for the system to use [42]. More than just reflective, our systems are "self-adaptive" [56]. That is, the collection of models also defines the operation of the system, so that changing the models changes the system. We believe that these self-adaptive systems are our best hope for reliable autonomy in complex environments.

Thus we generally work along the lines of [3] [51], but with significant differences in detail. We want as little as possible to be built-in and therefore fixed and unchangeable, and as much as possible to be modeled explicitly and therefore analyzable and modifiable. Ideally, everything is described using explicit models that are analyzable, which implies that self-analysis is possible. The reason we want these characteristics is simple: purely reactive agents are way too slow to operate effectively in complex environments; we need *anticipatory agents* [13], with models of their own processes, so they can identify anomalies in their behavior or in their environment's behavior [24].

The main hard problem is to define the connection between the models (descriptions) and the processes. This connection must be effective in both directions, from descriptions to processes (by interpretation), and from processes to descriptions (by model construction, analysis, and validation), though the system does not need to analyze those processes for which it is already given a description (besides, this is hard). We would, however, like to have some method of validation that the description does in fact describe the process (this is also hard in general).

Such a system may therefore need bootstrap processes that exist separately from the models and are not derived from them (at least at first), but we claim that this will not be necessary after the first implementations. The reason is that we can start with some method of converting process descriptions to processes (e.g., compilation or interpretation), and one of the descriptions describes that method. The original conversion program is written in some language for which previously existing conversion tools exist, or converted by hand into an executable program. After that first run, subsequent conversions are computed using the description of the conversion process and the process produced in the previous cycle.

The attempt to remove the bootstrap process entirely (thereby resulting an a so-called "pure" definition) has generated much heat and little light, but there are several new mathematical developments that lead to interesting approaches and possibilities. The most common response to the problem is capitulation, such as in [25], where multiple agents each examine the rest to avoid the circular self-analysis, and in all other approaches that separate the "watchers" from the "watched".

There are also many different kinds of arguments that it cannot be done, such as those in [13], referring to Löfgren's linguistic complementarity: "no language can be completely defined within itself" [47], and those in [21] that show that infinite regress or circularity in definitions is unavoidable. It turns out that the necessity of grounding all the definitions externally or otherwise, which is essential for the usual kind of inductive definitions [1], is not the only possible approach.

A very promising approach comes from Category Theory [48]. Co-inductive definitions can also be useful and much more powerful [23], do not need complete grounding, and it is shown in [19] that circularity in a set of definitions can be a powerful and well-defined computational tool. Similarly, well-foundedness is not necessary for meaningful definitions [30]. Non-well-founded Sets have a theory as good as (i.e., as consistent as, if not as well known as) ordinary Set Theory [2] [5], and can also be used in a co-inductive way to prove properties of systems.

Similarly, [47] describes the well-known tension between interpretability and describability: the more expressive a language is, the harder it is to interpret. We can avoid much of the difficulty if the symbol system (i.e., the language itself) can change, a possibility we have considered and advocated for some time [32] [36]. It does not change the fact that there is a trade-off; it merely moves it farther away. These Computational Semiotics issues are beyond the scope of this paper.

2 Computing

In this section, we describe some important activities in computing, and show how our earlier system engineering work reflects them.

Four important activities in computing systems are areas of active research:

- problem solving
- knowledge representation
- communication and other interactions
- mobility and transport

These are the most important basic computing activities, and new approaches to any of them can lead to much more effective system designs.

Problem solving involves algorithms, specifications, and other activity or action definition systems. Problem identification is also problem solving for the meta-problem “identify problem” (we have seen that these meta-problems arise frequently and naturally). Other examples are problem space definition and resource definition. There are many approaches to studying problem solving in humans [15] and in computing systems [18], but essentially none of them directly addresses the meta-problems.

Knowledge representation involves notations, formalisms, and other representational systems, active and passive. These notations can be systematic, computational, or formal, with increasing amounts of rigor and reliability in their definition, interpretation, and use. This area in one of the most active, since representation underlies all computing behavior. Since we expect to represent everything in a system, we need to go beyond the graph theoretical models [64] [9] [59] to include more of the modeling assumptions and ontological choices [60].

Communication and other interactions involve interface protocols and other communication mechanisms, with rules of discovery, engagement, information sharing, error recovery, and content interpretation. There has been great interest for some time in this area, spurred by the rise of distributed systems and the Internet, and there are some well-known formal modeling techniques [52] [22] that provide useful support for reasoning about communication aspects of programs. Of course, once the programs can communicate, it is an entirely different problem for them to interpret what the other one says in a useful way [54].

Mobility and transport involve protocols for movement and other transport mechanisms, with rules of connection establishment, navigation and re-establishment, and hand-off. This area has grown more recently, with the rise of wireless computing and portable systems, and the mathematical models are correspondingly newer [53] [55].

We have concentrated in the first two of these areas, problem solving and knowledge representation, and developed some approaches to Constructed Complex System development that can be viewed as the result of a meta-analytic approach. We describe how some of our research in "Integration Science" [6] [7] gathers these areas together in interesting ways. We provide few details here; much more information can be found in the references.

Our other studies in semiotics and symbol systems are an attempt to find representational mechanisms that are more flexible, so we can implement reflective autonomous systems that can improve their own behavior, not just by optimization in a fixed behavior space, but by changing that behavior space entirely [39] [42] [43]. Describing these applications would take us too far afield, though, so we concentrate on the most well-developed two results: Wrapping and Conceptual Categories. Both of them can be considered as the result of meta-analysis (this is their commonality).

Our Wrapping approach to integration infrastructure can be derived as a meta-analysis of problem solving, with an explicit identification and separation of all aspects (except perhaps inspiration) into particular interacting processes, and the use of Computational Reflection as a way to increase the power of the approach (e.g., "problem definition" is another problem that requires definition and interpretation).

Our Conceptual Categories can be derived as a meta-analysis of knowledge representation, with the awareness that not all knowledge is explained, some is just asserted. The operative principle is the separation of the role of some representational element from the role filler (this is also another instance of the Problem Posing Interpretation), reconnected through any of several mechanisms. We use the term "Conceptual Categories" in this regard because they are intended to model the categories and concepts of human language, and to distinguish them from the mathematical term "Category" from Category Theory [48] [63] [44], which we also use in some of our system descriptions. We describe Conceptual Categories a little later on, and start with Wrappings.

We have developed a knowledge-based approach to integration infrastructure for Constructed Complex Systems called "Wrappings" [33] [41], which is based on two key complementary parts: (1) explicit, machine-interpretable information ("meta-knowledge") about all of the uses all of the computational resources in a *Constructed Complex System* (including the user interfaces and other presentation mechanisms); and (2) a set of active integration processes that use that information to Select, Adapt, Assemble, Integrate, and Explain the application of these resources to posed problems. We have shown how the approach can be used to combine models, software components, and other computational resources into Constructed Complex Systems, and that it is a suitable implementation approach for complex software agents [35]. The process begins with our new interpretation of all programming languages that we have called the *Problem Posing Interpretation* [34].

2.1 Problem Posing Paradigm

All programming and modeling languages implement a notion of information service, with information service requests and information service providers. In most of these languages, the two of them are connected by using the same names [34], even though they can usefully be considered to be in different name spaces.

"Problem Posing" is a new declarative programming paradigm that unifies all major classes of programming. Programs interpreted in this style "pose problems"; they do not "call functions", "issue commands", "assert constraints", or "send messages" (these are information service requests). Program fragments are "resources" that can be "applied" to problems; they are not "functions", "modules", "clauses", or "objects" that do things (these are information service providers).

The Problem Posing Interpretation separates the requests from the providers. This is easy: compilers and interpreters always know the difference anyway, and so do we when we write the programs. We call the service requests "posed problems". We call the service providers "resources".

We can connect them through Knowledge Bases or by other means, but we have mainly considered the former, the "Knowledge-Based" Polymorphism that maps problems to resources, from the problem specification in its context to the computational resources that will organize the solution.

Any programming or modeling language can be interpreted in this new way.

The Problem Posing Interpretation changes the semantics of programming languages, not the syntax. It turns a program into an organized collection of posed problems, instead of an organized collection of solutions without problems. That should make programs easier to understand, because the problems at all levels of detail remain in it (we believe that part of the difficulty of debugging programs is that they are written as solutions without explicit statements of the corresponding problems).

Problem Posing also allows us to reuse legacy software with no changes at all, at the cost of writing a new compiler that interprets each function call, for example, not as a direct reference to a function name or address, but as a call to a new "Pose Problem" function, with the original function call as the specified problem and problem data. With this change from function calls to posed problems, the entire Wrapping infrastructure can be used as one important and powerful way to connect problems to resources. In particular, as the usage conditions for the legacy software change (which they always do), that information can be placed into the problem context, and used to divert the posed problems to new resources written with the new conditions in mind (only the timing characteristics will change, but those changes are frequently completely subsumed by using faster hardware). The gradual transition away from the legacy code is extremely important. Writing such a compiler is a well-understood process, and it is often worthwhile to do so.

The Problem Posing Interpretation radically changes our notion of autonomy, because it eliminates the notion of users "commanding" a system. It replaces that notion with the inclusion of users among the resources that can be used to address a problem. From this viewpoint, the more autonomous agents are merely the ones that need less help in deciding what to do, whether the decision is about choosing high-level goals or lower-level tasks that are expected to address previously determined goals.

This interpretation extends the separation of "what" and "how", the interface and the implementation, by adding the problem as a third level of abstraction, the "why", in addition to the resource interface and implementation. The usual kind of declarative program is a collection of solution statements, without any remaining record of the problems and sub-problems being addressed. Including the problems is a way to keep the motivation for algorithm decisions available.

The active part of mapping problems to resources can be implemented with tasking agents, but even in agent based systems [10] [17] [62], the tasking of agents is more of an assignment (or discussion) of what to do than a description of a problem to be solved. In some cases the presentation of a result to be obtained can be usefully viewed as a kind of problem posing, though it is not nearly as pervasive in those systems as our intentions would have it (which leads to some of the rigidity problems of migrating agents from one framework to another).

2.2 Wrapping

The Wrapping approach to integration infrastructure is particularly well-suited to the Problem Posing Paradigm. It can be derived from a meta-analysis of problem solving, with the awareness that not all problems are solved, some are just addressed. The principle is the separation of problem from resource (i.e., the Problem Posing interpretation), and their reconnection through the knowledge bases.

Wrappings not only emphasize meta-knowledge about the uses of computational resources, together with brokering and mediation of all component interactions (all critical concepts, as seen increasingly in other approaches), but it also regards as equally important the special resources for organizing and processing this information in a flexible and evolvable fashion. These algorithms are called *Problem Managers*, and they are used as a heartbeat to drive any system organized around posed problems.

The Wrapping approach, because it Wraps all of its resources, even the active integration processes, results in systems that are Computationally Reflective. That is, a system organized in this way has a machine-interpretable model of itself; the Wrapping resources and their interactions allow, in essence, a simulation of the entire system to be contained within the system. This allows sophisticated instrumentation and adaptive processing. It is this ability of the system to analyze and modify its own behavior that provides the power and flexibility of resource use. These ideas have proven to be useful, even when implemented and applied in informal, *ad hoc* ways.

The Wrapping theory has four fundamental properties that we regard as essential:

1. Everything in a system is a *resource* that provides an *information service*, including tools, components, interfaces, even architectures.
2. Every activity in a system is *problem study* in a particular *problem context.* All activities occur as a resource is *applied* to a *posed problem* in a particular *problem context*), including user interactions, information requests and announcements within the system, service or processing requests, and all other processing behavior. Problems to be studied are separated from the resources that might study them.
3. *Wrapping Knowledge Bases (WKBs)* contain *Wrappings*, which are explicit machine-interpretable descriptions of all of the resources in a system and how they can be applied to problems. Wrappings contain much more than "how" to use a resource. They also include both qualitative and quantitative information to help decide "when" it is appropriate to use it, "why" it might be used, and "whether" it can be used in this current problem and context.
4. *Problem Managers* (PMs) are the active integration processes that organize the use of resources. They are algorithms that use the Wrapping descriptions to collect and select resources to apply to problems. The system uses implicit invocation, both context- and problem-dependent. Resources are connected to problems via *Knowledge-Based Polymorphism.* These Wrapping processes are also resources; they are also Wrapped (Computational Reflection). The *Coordination Manager* (CM) and *Study Manager* (SM) are two important classes of PM.

The most important conceptual simplifications that the Wrapping approach brings to integration are the uniformities of the first two features: the uniformity of treating everything in the system as resources, and the uniformity of treating everything that happens in the system as a problem study. The most important algorithmic simplification is the reflection provided by treating the PMs as resources themselves: we explicitly make the entire system reflective by considering these programs that process the Wrappings to be resources also, and Wrapping them, so that all of our integration support processes apply to themselves, too. It is this ability of the system to analyze its own behavior that provides some of the power and flexibility of resource use, and that we believe is essential for effective autonomy in computing systems.

The key to all of this flexibility is the computational reflection that allows the system to make choices of computational resources at its very core; every resource, including the ones that make the choices, can be chosen, according to the posed problem at hand and the computational context in which the problem is being addressed. This kind of reflection is extremely important for autonomous systems [35] [38], since it allows them to assess their own behavior, and make changes in it as needed. Moreover, we have further developed this approach into a way to build systems that have complete models of their own behavior (at some level of granularity), that are used to generate that behavior, so that changing the models changes the behavior [42].

In summary, an integration infrastructure needs to put pieces together, so it needs the right pieces (resources and models of their behavior), the right information about the pieces (Wrapping Knowledge Bases), and the right mechanisms to use the information (Study Manager, Coordination Manager, and other Problem Managers). Because the system has access to its own computational processes, it can examine, assess, and change them.

2.3 Conceptual Categories

The most well developed modeling technique that we use for knowledge representation is our notion of "Conceptual Categories" [37] [40], which is a very flexible knowledge representation technique that includes the viewpoints and modeling choices made, so that different kinds of models of the same concepts can be compared, and so that the system can access its own modeling decisions for evaluation and / or adjustment.

Its importance for modeling complex systems is that it allows us to keep track of the very different interests of the different stakeholders, from operators and planners to technology developers, acquisition organizations, and even policy makers. Each of these groups has a very different interaction with these systems, at different time scales and for different purposes.

We want to use Conceptual Categories because they allow us as designers to retain and compare many of the modeling decisions, so that we can explore different choices. As the development of a system design matures, we can use this mechanism more widely (to model more of the system components) and more deeply (to model the components at greater detail).

The Conceptual Category approach is based on modeling all concepts involved in a system, and separating the structure of these concepts (the *divisions*) from the fact of their existence (the *categories*). Then we can connect the categories to different structures using many different mechanisms. Conceptual Categories are intended to model our notion of categories of concepts [40]. They generalize ordinary set theory in four directions: our collective objects

- have indefinite boundaries;
- have indefinite elements;
- allow leakage from context; and
- contain elements that may have multiple alternative structures.

A category is the representation of a concept, whether a word or a phrase (for simplicity, in this description we consider only American English as the relevant language). The connotations of the phrase depend on the viewpoint taken by whatever individual or system is using it. The divisions are used to maintain the different structures according to the different viewpoints.

For each category, there is one or more *divisions*, each of which is a structure-defining mechanism (partition into subcategories, decomposition into parts, role as a part or partition of some other category, etc.). The interpretation system, when asked for information or processing about an element in the category, chooses which division to use according to an associated *viewpoint*, which is a description of the focus of the concept (the scope or extent, the scale or level of detail, and assumptions), and its context (which provides all interpreters of symbolic expressions that use the concept).

We showed before that our Conceptual Categories [40] offer a new way to combine many modeling methods, including both declarative and procedural knowledge, into a single coherent framework, by (1) separating the representations from the interpretations, so that different interpreters can be chosen in different contexts, and different construction methods can be used; (2) making all of the interpreters explicit, so that they can be compared and analyzed, and new ones more easily integrated; and (3) allowing alternative conceptual representation methods in the same system.

We showed that this approach provides several advantages for flexible modeling: (1) an explicit place for context and viewpoint in the knowledge structure; (2) an explicit treatment of modeling assumptions; (3) multiplicity of constituent structures according to viewpoint (context-dependent flexibility); (4) multiple interpreters and other procedures for the same declarative knowledge, selected according to context (situated knowledge); and (5) a Computationally Reflective knowledge representation [38]: our descriptions of the terms and concepts of Conceptual Categories are naturally represented as categories.

This approach promises to be the most appropriately general for defining the knowledge available to a system.

3 The Separation Principle

The principle that both of these applications have in common is separation, which is the method we use to gain flexibility in system design, development, implementation, and execution.

Any approach to flexibility in systems makes some fixed relationships variable, that is, flexibility breaks those relationships (which are usually implicit because they are fixed), which implies separating otherwise conflated elements (and therefore finding some difference criterion). Any such separation implies that there must also be coordination methods that can put the separate pieces back together, but because the coordination methods are separated from the pieces, many alternative methods can be used. This separation principle leads to great flexibility and power [6] [41].

For example, we can take each programming language to be a computational model (usually a fairly simple one), together with a lot of peripheral methods that are needed for interactions with the host operating system. The reason this decomposition is interesting to us is that almost all of these peripheral methods are essentially the same across many programming languages, regardless of computational model. It seems useful to us to use the same language parts as much as possible, instead of having different parts for different languages just to do so. The requisite interconnection infrastructure can be defined using Wrappings.

We present another example from the representation activity in much more detail: the meaningfulness hierarchy called DIKU: the separation of the generic terms information and knowledge into more specific and precise notions.

3.1 Meaningfulness Hierarchy

Our version of the hierarchy of meaningfulness seems to be a "folk classification", though the idea for it apparently began in a poem of T. S. Eliot:

> Where is the wisdom we have lost in knowledge?
> Where is the knowledge we have lost in information?
> (The Rock, I (1934))

While this observation about the loss of meaning was originally only vaguely related to the subject of this paper, it seems to have evolved over time into the DIKU sequence (Data, Information, Knowledge, Understanding), which can be described as follows [29] (we leave Wisdom for later, since we have no idea how to define it operationally):

- Data is the bits, the raw binary distinctions.
- Information is data with an interpretive context.
- Knowledge is information with a knower.
- Understanding is knowledge in perspective.

Our interest in this hierarchy is in the important transformation steps that increase meaningfulness, moving external phenomena to data and further up into more meaningful (and more humanly useful) forms.

The phenomena are whatever it is that we intend to study. They are often out in the real world, in which case we only have descriptions (i.e., models) to work with.

Data is the bits, the raw binary distinctions, computed according to our modeling assumptions and conventions, and represented as symbols in some symbol system via some representational mechanism.

> "There is no data without a model"
> (We have not been able to find the source of this quote)

The phenomena are not the same as the data extracted from them ("The map is not the territory"). It is important to record and remember the modeling assumptions that go into the extraction of data from some external phenomenon, even for basic measurements, since it is often the anomalies in these basic measurements that lead to advances in the science. Noise processes and models thereof are especially important for systems expected to operate in complex environments [4].

Information is data with an interpretive context. Different interpreters may occur for different contexts, so different information may occur for the same data. A simple case of this distinction is in the 32-bit computer words, that can be either fixed point or floating point numbers (or other things), depending on how they are interpreted, and the preferred interpretation is not contained in those 32 bits. It is for this reason that we emphasize the context of any data, and indeed meta-knowledge about all kinds of data, to guide the interpretation.

Knowledge is information with a knower, that is, an entity with purposes or intentions for the use of the information. It may be combined in different ways to different ends. Information may be adapted to different ends by different purposes. In our opinion, therefore, there is no such thing as abstract knowledge, out of context, out of purpose (though we do not intend to argue the point here, except in terms of computing systems).

Understanding is knowledge in perspective, across multiple applications and in multiple contexts. This is the most difficult of our meaningfulness levels for computing systems (excepting wisdom, which we altogether avoid as being much too hard to consider for computing systems), and we have only a few preliminary notions about it.

There is a set of transformations within and between all levels of the hierarchy. We concentrate on the levels closer to the external phenomena, because they are the most often used and most easily justified, but we mention some of the difficulties at the higher levels.

We discuss these transformations in a hierarchy of structuring spaces, with varying amounts of implicit structure, and various assumptions about what is being represented (knowledge representation mechanisms are mappings from knowledge into data, even as we use data to represent information under a usually implicit context for interpretation).

3.2 Phenomena to Data

This step is the first transformation; from whatever it is out in the external world to computable data. This transformation encompasses what is usually called “modeling”, that is, the mapping of the phenomenon of interest to some formal space, even if the formal space has only a very simple structure. The modeling steps are essentially the same as those we listed before:

1. identification of the phenomenon of interest and its important properties;
2. selection of a formal space that may reflect those properties;
3. mapping the phenomenon into the formal space;
4. deductions and other operations within that formal space;
5. remapping the results back to the phenomenon.

The reason we emphasize all of these steps is that most descriptions of modeling mention only the third and fourth steps. without noting that the formal space is to be chosen for convenience and fidelity, and especially without noting how steps (1), (3), and (5) interact.

These modeling transformations are the semiotic gatekeepers; they are used to extract symbols from the real world phenomenon (or any other phenomenon; not all models are models of the real world). One of the difficulties of these mappings is their two conflicting goals: to extract as much information as possible from the environment, while not producing structures that cannot be interpreted quickly enough.

Along this line, we have proved two theorems that we called "Get Stuck" theorems [36], that are limitations on what symbol systems can do.

The first of these theorems is about the expressive power of symbol systems. For any fixed finite symbol system (that is, a fixed finite set of basic symbols and a fixed finite set of combination rules), the number of expressions of any length can be counted (using simple generating functions from combinatorial theory). The result is that expressing more and more detailed distinctions requires longer and longer expressions, and eventually they get too long to process in a timely manner. That is, the system "gets stuck" in its expressive capabilities. The second theorem is about knowledge structures, and shows a similar limitation, even when humans are adding the knowledge elements.

These problems are fundamental to any data representation mechanism, and precede even those that transform data into other data.

3.3 Data to Data

Most computation is transformation of data to data. We will limit ourselves to a few remarks about this kind of transformation, since many other papers address it. Computers can do only two things with data (we are speaking of commodity digital computers):

- copy it
- compare it

They can also interpret a small fixed set of fixed finite models of other formal theories. For example, and we cannot emphasize this too strongly, computers do not "do arithmetic"; they interpret two fixed finite models of arithmetic (one for fixed point and one for floating point). This limitation is why we must map everything into data.

Each of the different kinds of data transformation (abstraction, aggregation, generalization, specialization, and others) needs to be considered in the context of the modeling assumptions that lead to the original data, and we expect that with that knowledge, we can produce some new relationships among these transformations. This research project is underway.

3.4 Data to Information

Most of our uses of computing are about transforming data or information to other information. One of the important implicit assumptions underlying the use of computing is that data relates to information (since the computers can only process data). It is our common assumption that the data does represent information, and it is a common failing of computing systems not to make that relationship explicit. Transformations that change data may or may not change the information (depending on the context).

This Section is about adding meaning to data that transforms it into information by giving it an interpretive context. Symbol interpretation turns data into information (semiotics), but that is not yet knowledge or meaning (semantics is discussed later on); they need some further additive transformations. These are the focus of another branch of our research.

Of course, the interpretive context is knowledge that is used to interpret the data (that is the system's purpose for the context), so we have a potentially recursive structure. Our Conceptual Categories are an appropriate data structure for representing knowledge in the framework we have described. This mechanism also retains the modeling assumptions in the structure, where they can be analyzed.

3.5 Other Transformations

The other transformations, from information to knowledge, and from knowledge to understanding, are more difficult to describe because of a lack of adequate operational definitions of the two terms.

Conceptual Categories are intended as a representation mechanism for both knowledge and understanding, and our separation of functions is intended to make them more easily computed. That means that we intend to map those two levels into information (data with an interpretive context) that can be analyzed computationally. This approach is also how we intend to map semantics into information, so that we can have more meaningful computational processes.

4 Conclusions

In a sense, all of these meta-analyses are just careful modeling, with a mathematically tuned eye for identifying assumptions and separating elements or relationships according to differences in external use as well as internal structure and / or behavior. The abstractions, notations and formalisms, and even the analytical frameworks, are all ways to find appropriate formal spaces in which to map the computing problems of interest, so that we can abstract the knowledge and understanding across disciplinary boundaries, and build much more interesting systems.

A healthy dose of mathematical aptitude and attitude is very helpful in this regard, since the identification of implicit assumptions requires an identification of those context elements and conventions that allow the assumptions to be left implicit, and the identification of appropriate abstractions requires an identification of the important relationships and interactions.

We have apparently been working in meta-informatics for many years.

References

1. Peter Aczel, "Inductive Definitions", Chapter C.7, pp. 739-782 in Jon Barwise (ed.), *Handbook of Mathematical Logic*, Studies in Logic and the Foundations of Mathematics, Volume 90, North-Holland (1977)
2. Peter Aczel, *Non-well-founded Sets*, CSLI Lecture Notes Number 14, Center for the Study of Language and Information, Stanford U., U. Chicago Press (1988)
3. James S. Albus, Alexander M. Meystel, *Engineering of Mind: An Introduction to the Science of Intelligent Systems*, Wiley (2001)
4. Fahiem Bacchus, Joseph Y. Halpern, Hector J. Levesque, "Reasoning about noisy sensors and effectors in the situation calculus", *Artificial Intelligence Journal*, Volume 111, pp. 171-208 (July 1999)
5. Jon Barwise, Lawrence Moss, *Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena*, CSLI Lecture Notes Number 60, Center for the Study of Language and Information, Stanford U. (1996)
6. Kirstie L. Bellman, Christopher Landauer, "Integration Science is More Than Putting Pieces Together", in *Proceedings of the 2000 IEEE Aerospace Conference (CD)*, 18-25 March 2000, Big Sky, Montana (2000)
7. Kirstie L. Bellman, Christopher Landauer, "Towards an Integration Science: The Influence of Richard Bellman on our Research", *Journal of Mathematical Analysis and Applications*, Volume 249, Number 1, pp. 3-31 (2000)
8. Richard Bellman, P. Brock, "On the concepts of a problem and problem-solving", *American Mathematical Monthly*, Volume 67, pp. 119-134 (1960)
9. Ronald J. Brachman, "What's in a Concept: Structural Foundations for Semantic Networks", *International Journal for Man-Machine Studies*, Volume 9, pp. 127-152 (1977)
10. Jeffrey M. Bradshaw (ed.), *Software Agents*, AAAI Press (1997)
11. Thomas A. Bruce, *Designing Quality Databases with IDEF1X Information Models*, Dorset House (1992)
12. E. W. Dijkstra, *A Discipline of Programming*, Prentice-Hall (1976)
13. Bertil Ekdahl, Eric Astor, Paul Davidsson, "Towards Anticipatory Agents", pp. 191-202 in M. Wooldridge, N. R. Jennings (eds.), *Intelligent Agents - Theories, Architectures, and Languages*, Springer LNAI 898 (1995); also on the Web at URL "http: //citeseer. nj. nec. com/ ekdahl95towards. html" (availability last checked 07 February 2004)
14. David W. Embley, Barry D. Kurtz, Scott N. Woodfield, *Object-Oriented Systems Analysis*, Yourdon Press (1992)
15. K. Anders Ericsson, Reid Hastie, "Contemporary Approaches to the Study of Thinking and Problem Solving", Chapter 2, pp. 37-79 in Robert J. Sternberg (ed.), *Thinking and Problem Solving*, Academic Press (1994)
16. Hans-Erik Eriksson, Magnus Penker, *UML Toolkit*, Wiley (1998)
17. Jacques Ferber, *Multi-Agent Systems*, Addison Wesley Longman (1999); translation of Jacques Ferber, *Les Systèmes Multi-Agents: Vers une intelligence collective*, InterEditions, Paris (1995)
18. Kenneth D. Forbus, Johann de Kleer, *Building Problem Solvers*, A Bradford Book, MIT Press (1993)
19. Joseph Goguen, Kai Lin, and Grigore Roşu, "Circular Coinductive Rewriting", pp. 123-131 in *Proceedings of ASE'00: The 15th International Conference on Automated Software Engineering*, 11-15 September 2000, Grenoble, France, IEEE Press (2000)

20. Joseph A. Goguen, Grant Malcolm, *Algebraic Semantics of Imperative Programs*, MIT Press (1996)
21. Francis Heylighen, "Advantages and Limitations of Formal Expression"; on the Web at URL "http: //citeseer. nj. nec. com/ heylighen99advantages. html" (availability last checked 07 February 2004)
22. C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall (1985)
23. Bart Jacobs, Jan Rutten, "A Tutorial on (Co)Algebras and (Co)Induction", *EATCS Bulletin*, Volume 62, pp. 222-259 (1997)
24. Catriona M. Kennedy, "A Conceptual Foundation for Autonomous Learning in Unforeseen Situations", Tech. Rpt. WV-98-01, Dresden Univ. Technology (1998); also on the Web at URL "http: //citeseer. nj. nec. com/ kennedy98conceptual. html" (availability last checked 07 February 2004)
25. Catriona M. Kennedy, "Distributed Reflective Architectures for Adjustable Autonomy", in David Kortenkamp, Gregory Dorais, Karen L. Myers (eds.), *Proceedings of IJCAI-99 Workshop on Adjustable Autonomy Systems*, 1 August 1999, Stockholm, Sweden (1999); a similar note is on the web at URL "http: //citeseer. nj. nec. com/ kennedy99distributed. html" (availability last checked 07 February 2004)
26. Gregor Kiczales, Jim des Rivières, Daniel G. Bobrow, *The Art of the Meta-Object Protocol*, MIT Press (1991)
27. Haim Kilov, James Ross, *Information Modeling*, Prentice-Hall (1994)
28. C. Kreitz, "Formal Mathematics for Verifiably Correct Program Synthesis", J. Interest Group Pure and Applied Logic, Volume 4, Number 1 (1996) on the Web at URL "http: //www. dcs. kcl. ac. uk/ journals/ igpl/ IGPL/ V4-1/" (availability last checked 07 February 2004)
29. Christopher Landauer, "Data, Information, Knowledge, Understanding: Computing Up the Meaning Hierarchy", pp. 2255-2260 in *Proceedings of SMC'98: The 1998 IEEE International Conference on Systems, Man, and Cybernetics*, 11-14 October 1998, San Diego, California (1998)
30. Christopher Landauer, Kirstie L. Bellman, "Mathematics and Linguistics", pp. 153-158 in Alex Meystel, Jim Albus, R. Quintero (eds.), *Intelligent Systems: A Semiotic Perspective*, *Proceedings of the 1996 International Multidisciplinary Conference*, *Volume I: Theoretical Semiotics*, *Workshop on New Mathematical Foundations for Computer Science*, 20-23 October 1996, NIST, Gaithersburg, Maryland (1996)
31. Christopher Landauer, Kirstie L. Bellman, "Model-Based Simulation Design with Wrappings", pp. 169-174 in *Proceedings of OOS'97: The 1997 Object Oriented Simulation Conference*, *WMC'97: The 1997 SCS Western MultiConference*, 12-15 January 1997, Phoenix, SCS International (1997)
32. Christopher Landauer, Kirstie L. Bellman, "Situation Assessment via Computational Semiotics", pp. 712-717 in *Proceedings ISAS'98: the 1998 International MultiDisciplinary Conference on Intelligent Systems and Semiotics*, 14-17 September 1998, NIST, Gaithersburg, Maryland (1998)
33. Christopher Landauer, Kirstie L. Bellman, "Generic Programming, Partial Evaluation, and a New Programming Paradigm", Paper etspi02 in *32nd Hawaii Conference on System Sciences*, *Track III: Emerging Technologies*, *Software Process Improvement Mini-Track*, 5-8 January 1999, Maui, Hawaii (1999); revised and extended version in Christopher Landauer, Kirstie L. Bellman, "Generic Programming, Partial Evaluation, and a New Programming Paradigm", Chapter 8, pp. 108-154 in Gene McGuire (ed.), *Software Process Improvement*, Idea Group Publishing (1999)

34. Christopher Landauer, Kirstie L. Bellman, "Problem Posing Interpretation of Programming Languages", Paper etecc07 in *Proceedings of HICSS'99: the 32nd Hawaii Conference on System Sciences*, *Track III: Emerging Technologies*, *Engineering Complex Computing Systems Mini-Track*, 5-8 January 1999, Maui, Hawaii (1999)
35. Christopher Landauer, Kirstie L. Bellman, "New Architectures for Constructed Complex Systems", in *The 7th Bellman Continuum*, *International Workshop on Computation, Optimization and Control*, 24-25 May 1999, Santa Fe, NM (1999); (to appear) in *Applied Mathematics and Computation*, Volume 120, pp. 149-163 (May 2001)
36. Christopher Landauer, Kirstie L. Bellman, "Symbol Systems in Constructed Complex Systems", pp. 191-197 in *Proceedings of ISIC/ISAS'99: The 1999 IEEE International Symposium on Intelligent Control*, 15-17 September 1999, Cambridge, Massachusetts (1999)
37. Christopher Landauer, Kirstie L. Bellman, "Relationships and Actions in Conceptual Categories", pp. 59-72 in G. Stumme (Ed.), *Working with Conceptual Structures - Contributions to ICCS 2000*, *Auxiliary Proc. ICCS'2000: The 8th Intern. Conf. on Conceptual Structures*, 14-18 August 2000, Darmstadt, Shaker Verlag, Aachen (August 2000)
38. Christopher Landauer, Kirstie L. Bellman, "Reflective Infrastructure for Autonomous Systems", pp. 671-676, Vol. 2 in *Proc. EMCSR'2000: The 15th European Meeting on Cybernetics and Systems Research*, *Symposium on Autonomy Control: Lessons from the Emotional*, 25-28 April 2000, Vienna (April 2000)
39. Christopher Landauer, Kirstie L. Bellman, "Some Measurable Characteristics of Intelligence", Paper WP 1.7.5, *Proceedings of SMC'2000: The 2000 IEEE International Conference on Systems, Man, and Cybernetics (CD)*, 8-11 October 2000, Nashville Tennessee (2000)
40. Christopher Landauer, Kirstie L. Bellman, "Conceptual Modeling Systems: Active Knowledge Processes in Conceptual Categories", pp. 131-144 in Guy W. Mineau (Ed.), *Conceptual Structures: Extracting and Representing Semantics*, *Contributions to ICCS'2001: The 9th Intern. Conf. on Conceptual Structures*, 30 July-03 August 2001, Stanford University (August 2001)
41. Christopher Landauer, Kirstie L. Bellman, "Wrappings for One-of-a-Kind System Development", Paper STSSV04 in *Proceedings of HICSS'02: The 35th Hawaii International Conference on System Sciences (CD)*, *Track IX: Software Technology*, *Advances in Software Specification and Verification Mini-Track*, 7-10 January 2002, Waikoloa, Hawaii (Big Island) (2002)
42. Christopher Landauer, Kirstie L. Bellman, "Self-Modeling Systems", pp. 238-256 in R. Laddaga, H. Shrobe (eds.), "Self-Adaptive Software: Applications", Springer Lecture Notes in Computer Science, Vol. 2614 (2003)
43. Christopher Landauer, Kirstie L. Bellman, "Refactored Characteristics of Intelligent Computing Systems", *Proceedings of PERMIS'2002: Measuring of Performance and Intelligence of Intelligent Systems*, 13-15 August, NIST, Gaithersburg, Maryland (2002)
44. F. William Lawvere, Stephen H. Schanuel, *Conceptual Mathematics: A first introduction to categories*, Cambridge (1997, reprinted with corrections 2000)
45. Ákos Lèdeczi, Árpád Bakay, Miklós Maróti, "Model-Integrated Embedded Systems", pp. 99-115 in [56]

46. Ákos Lèdeczi, Árpád Bakay, Miklós Maróti, Péter Völgyesi, Greg Nordstrom, Jonathan Sprinkle, Gábor Karsai, "Composing Domain-Specific Design Environments", *IEEE Computer*, Volume 34, Number 11, pp. 44-51, Novermber 2001 (2001)
47. Lars Löfgren, "Phenomena of Autonomy with Explanations in Introspective Language"; on the Web at URL "http: //citeseer. nj. nec. com/ 176143. html" (availability last checked 07 February 2004)
48. Saunders MacLane, Garrett Birkhoff, *Algebra*, Macmillan (1967)
49. Pattie Maes, "Concepts and Experiments in Computational Reflection", pp. 147-155 in *Proceedings OOPSLA '87* (1987)
50. P. Maes, D. Nardi (eds.), *Meta-Level Architectures and Reflection*, *Proceedings of the Workshop on Meta-Level Architectures and Reflection*, 27-30 October 1986, North-Holland (1988)
51. Alexander M. Meystel, James S. Albus, *Intelligent Systems: Architecture, Design, and Control*, Wiley (2002)
52. Robin Milner, *A Calculus of Communicating Systems*, SLNCS 92, Springer (1980)
53. Robin Milner, *communicating and mobile systems: the π-calculus*, Cambridge (1999)
54. Andrea Omicini, Franco Zambonelli, Matthias Klusch, Robert Tolksdorf (eds.), *Coordination of Internet Agents: Models, Technologies, and Applications*, Springer (1998)
55. Gordon Plotkin, Colin Stirling, Mads Tofte (eds.), *Proof, Language, and Interaction: Essays in Honor of Robin Milner*, MIT Press (2000)
56. Paul Robertson, Howie Shrobe, Robert Laddaga (eds.), *Self-Adaptive Software*, SLNCS 1936, Springer (2001)
57. Bertrand Russell, *The Scientific Outlook*, George Allen and Unwin, London; W. W. Norton, New York (1931)
58. Mary Shaw, William A. Wulf, "Tyrannical Languages *still* Preempt System Design", pp. 200-211 in *Proceedings of ICCL'92: The 1992 International Conference on Computer Languages*, 20-23 April 1992, Oakland, California (1992); includes and comments on Mary Shaw, William A. Wulf, "Toward Relaxing Assumptions in Languages and their Implementations", *ACM SIGPLAN Notices*, Volume 15, No. 3, pp. 45-51 (March 1980)
59. John F. Sowa, *Principles of Semantic Networks: Exploration in the Representation of Knowledge*, Morgan Kaufmann (1991)
60. John F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks/Cole (2000)
61. Guy L. Steele, Jr., Gerry J. Sussman, "The Art of the Interpreter, or, the Modularity Complex", (Parts Zero, One, and Two), AI Memo 453, MIT (1978)
62. V. S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, Robert Ross, *Heterogeneous Agent Systems*, MIT Press (2000)
63. R. F. C. Walters, *Categories and Computer Science*, Cambridge Computer Science Texts, Cambridge (1991)
64. William A. Woods, "What's in a Link: The Semantics of Semantic Networks", pp. 35-79 in D. G. Bobrow, A. M. Collins (eds.), *Representation and Understanding*, Academic Press (1975)

A Dynamic Aspect Weaver over the .NET Platform

Luis Vinuesa and Francisco Ortin

University of Oviedo, Computer Science Department
Calvo Sotelo s/n, 33007 Oviedo, Spain
{vinuesa,ortin}@lsi.uniovi.es
http://www.di.uniovi.es/~{vinuesa,ortin}

Abstract. Since *Aspect Oriented Software Development* (AOSD) has appeared, a set of tools have been developed that enable its use at software development. On many occasions, applications must be capable of adapting to runtime emerging requirements. This is possible through the use of *dynamic weaving* tools. However, these tools are subject to limitations such as language dependency or a limited set of join points, which limit their utility.

This paper shows the research we are carrying out in the dynamic weaving field. We apply *computational reflection* at the *virtual machine* computational level in order to achieve dynamic weaving regardless of the language. We have selected the .NET system as the platform to develop our dynamic weaver, making possible the dynamic adaptation of platform-independent applications.

1 Introduction

In many cases, significant concerns in software applications are not easily expressed in a modular way. Examples of such concerns are transactions, security, logging or persistence. The code that addresses these concerns is often spread out over many parts of the application. It is commonly tangled with the application logic and, therefore, it is more difficult to maintain, understand and reuse.

In the Software Engineering area, the principle of *separation of concerns,* SoC [1],[2] has been used to manage the complexity of software development; it separates main application algorithms from special purpose concerns (typically orthogonal to the main functionality) – e.g. authentication, logging, memory management, performance, etc – during design phase and implementation (coding) phase. Final applications are built by means of their main functional code plus their specific problem-domain concerns. The main benefits of this principle are:

1. Higher level of abstraction - the programmer can reason about individual concerns in isolation.
2. Easier understanding of the application functionality - the application's source code is not cluttered with the other concerns' code.
3. Concern reuse - SoC attains decoupling of different modules, achieving reusability of single concerns.
4. Increase of application development productivity - in addition to the advantages previously mentioned, the use of testing and debugging concerns (such as tracing, pre and post condition contract enforcement or profiling) might facilitate the application construction, without modifying the functional source code.

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 197–212, 2004.

This principle has been performed following several approaches. Multi-dimensional separation of concerns [3], Composition Filters [4], or the more recent Aspect Oriented Software Development (AOSD) [5] are well-known examples.

AOSD is a development methodology whose main aim is the construction of concern-adaptable programs. Most existing tools lack adaptability at runtime: once the final application has been generated (*woven*), it will not be able to adapt its concerns (aspects) at runtime. This is known as *static aspect weaving*. There are certain cases in which the adaptation of application concerns should be done dynamically, in response to changes in the runtime environment – e.g., distribution concerns based on load balancing [6]. Another example of using dynamic AOSD is what has been recently called *autonomic software*: software capable of being itself repaired, managed, optimised or recovered.

To overcome the static-weaving tools limitations, different dynamic-weaving AOP approaches – like PROSE [7], Dynamic Aspect-Oriented Platform [8], JAC [9], AOP/ST [10] or CLAW [11]- have appeared. However, as we will explain in next sections, they limit the set of join points they offer (with respect to static weaving tools), restricting the way applications can be adapted at runtime. As well, they use fixed programming languages, therefore aspects and concerns are not reusable out of its own programming language.

Reflection is a programming language technique that achieves dynamic application adaptability. It can be used to get aspect adaptation at runtime. Most runtime reflective systems are based on the ability to modify the programming language semantics while the application is running (e.g., the message passing mechanism).

As a previous research work, we have developed a system, called nitrO [12], which offers non-restrictive computational reflection in a language-neutral way, and might be used as an AOSD front-end. By using a virtual-machine-based platform, any program can modify any other program (apart from modifying itself) at runtime regardless the language and with no restrictions.

This prototype presented some drawbacks, such as its runtime performance – due to the interpretation of every application, and the use of a highly adaptable reflection technique –, as well as the requirement of defining any language grammar to be used by its applications.

Afterwards, the language and platform independent Microsoft .NET platform appeared [13]. We are applying the concepts we have learned to this platform which offers both a good runtime performance and language and hardware neutrality. We are developing a reflective treatment of .NET applications offering a dynamic weaving framework.

When a compiled application is going to be executed, the system transforms its code (manipulating its *Microsoft Intermediate Language,* MSIL) to allow other applications to access it and adapt it. To do that, the system has a server (application registry) in which every application that is being executed is registered. Once an application is running, if another one needs to adapt it, the server will offer any running application adaptation following the AOSD principle.

The rest of this paper is structured as follows. In the next section we present AOSD and its current research situation. Section 3 presents dynamic weaving and the main lacks of the existing tools that offer it. Then in section 4 we present a classification of reflection; the benefits we obtain using reflection in abstract machines are presented in section 5. In section 6 we present our system structure and then its runtime performance is analysed. Section 8 presents the benefits we obtain with our system and the final conclusions are presented in section 9.

2 Aspect Oriented Software Development (AOSD)

In the early days of computer science, data and code were tangled in complex structures – that is what was called *spaghetti code* [14]. Then came functional decomposition and structured languages allowing us to build more complex and modular software. However, as software complexity grew, we needed better techniques, such as object-oriented programming (OOP), which let us view a system as a set of collaborating objects.

In many cases, in software applications we have to deal with some concerns that are orthogonal and independent to the main concern, and sometimes reusable. Examples of such concerns are synchronization, error management, authentication, memory management, security, logging or persistence. These concerns usually spread over the different modules of the application. OOP has shown its strength when it comes to modelling common behaviour; however, OOP does not adequately address orthogonal concerns that span over many, often unrelated, modules.

Aspect Oriented Software Development, AOSD, [5] arises to fill this void. The Aspect-Oriented Programming method [5] provides explicit language support for modularising application concerns that crosscut the application functional code. Aspects express functionality that cuts across the system in a modular way, thereby allowing the developer to design a system out of orthogonal concerns and providing a single focus point for modifications. By separating the application functional code from its crosscutting aspects, the application source code would not be tangled, being easy to debug, maintain and modify [1].

Aspect-oriented tools create programs combining the application functional code and its specific aspects. The process of integrating the aspects into the main application code is called *weaving* and a tool called *aspect weaver* performs it. This weaving process can be done in a static or dynamic way.

2.1 Static Weaving

Most current AOP implementations are largely based on static weaving: compile-time modification of application source code, inserting calls to specific aspect routines. The places where these calls may be inserted are called *join points.*

AspectJ [15],[16] is an example of a static-weaving aspect-oriented tool: a general-purpose aspect-oriented extension to Java that supports aspect-oriented programming. An AspectJ application is composed of:

- *Join points:* well defined execution points in a program's execution flow. For example, joinpoints could define calls to specific methods in a class or read/write access to a field.
- *Pointcuts:* a pointcut selects join points, which are described by the *pointcut declaration.*

- *Advice:* code that runs at each join point selected by a pointcut. This code will be executed when a join point is reached, either before or after the computation proceeds.
- *Aspect:* a modular unit of crosscutting implementation that is provided in terms of pointcuts and advice, specifying *what* (advice) and *when* (pointcut) its code is going to be executed.

In order to generate the final application the AspectJ compiler "`ajc`" [17] takes both the application functional code and its specific aspects, producing the final Java2 ".class" files.

Static weaving is subject to a number of drawbacks such as the impossibility of adapting in response to changes on the runtime environment, since the final application is generated weaving core functionality with aspects at compile time. Any adaptation needed at runtime forces the execution to stop; then the application is recompiled and restarted. That is the reason why static weaving is not suitable for use in non-stop applications, which may need to be adapted.

At the same time, debugging an *aspected* application is a hard and complex task because the code you must analyse is the result of the weaving process: the core code is tangled with the aspects code, which may be scattered all over the application.

As we comment in the next section, the final disadvantage is that these tools are mostly language dependent – e.g., AspectJ only supports the Java programming language.

2.2 Language Dependency

An important drawback of most of the existing tools – either dynamic or static ones – is that they use a fixed language. Both aspects and core functionality must be implemented in its unique programming language. This is an important restriction, because all applications written in other languages are excluded from the possibility of being adapted, and we cannot develop aspects in other languages.

In order to overcome this drawback some language independent systems based on .NET platform have appeared: e.g. CLAW [11], Weave.NET [18], LOOM.NET [19][20][21] and Dechow's proposal [22]. These systems are currently in a development stage and there are no binaries for them (except LOOM.NET, which has published some preliminary versions).

These systems achieve language independency by means of doing all the work at intermediate language (IL) level, without the need of the source code, making possible to weave aspects written in any language with applications written in any other language.

Weave.NET and Dechow's proposal are static systems, doing all the work at compile or load time. LOOM.NET has two different parts, one is a static weaver and the other is a dynamic weaver. CLAW is dynamic.

3 Existing Dynamic-Weaving Tools

Using a static weaver, the final program is generated by weaving the application functional code and its selected aspects. If we want to enhance the application with a new aspect, or even eliminate one of the aspects woven, the system has to be recompiled and restarted.

Although not every application aspect needs to be adapted at runtime, there are specific aspects that will benefit from a dynamic-weaving system. There could be applications that need to dynamically adapt their specific concerns in response to changes in the runtime environment [7]. As an example, related techniques have been used in handling Quality of Service (QoS) requirements in CORBA distributed systems [23] or in managing web cache prefetching [24].

In systems that use dynamic weaving aspects, core functionality remains separated in all the software lifecycle, including system execution. Resulting code is more reusable and adaptable, and aspects and core functionality can evolve independently [8].

In order to overcome the static-weaving weaknesses, different dynamic-weaving approaches have emerged: e.g. AOP/ST[10], PROSE [7], Dynamic Aspect-Oriented Platform (DAOP) [8], Java Aspects Components (JAC) [9], CLAW [11], LOOM.NET (its dynamic part)[21]. These systems offer the programmer the ability to dynamically modify the aspect code assigned to application join points – similar to runtime reflective systems based on *meta objects protocols* (MOP) [25].

Existing systems have some drawbacks:

1. They offer a limited set of language join-points restricting the amount of application features an aspect can adapt. For instance, PROSE cannot implement a post-condition-like aspect, since its join-point interface does not allow accessing the value returned by a method upon exit [7]. JAC allows only three different aspects: *wrap, role and exception handler*, so accessing a field can not be controlled [9]. LOOM.NET only allows virtual methods to be dynamically interwoven; other members of a class, such as fields, properties, static, and class functions currently can not be accessed this way [21].
2. The use of a fixed language is another important limitation of the majority of existing systems. JAC and PROSE are implemented over Java, and applications and aspects can only be developed in Java. This limits the number of applications that can use aspects, and also limits the reusability of those aspects because they can't be used in another language. Other tools like DAOP fix their own aspect definition language [8], so they have the same limitations mentioned before.
3. Some of the systems that claim to be dynamic need aspects to be defined at design time, and they are instantiated at runtime. This offers some of the benefits of dynamic weaving but makes impossible to use those tools with applications that are running and have been designed without aspects. As an example, in DAOP you must define at design time the system architecture, describing aspects and components interfaces and their connection. In CLAW [11] you must use a set of weave definition files describing how join points must be mapped to an aspect; each join point is identified selecting the portion of the target method to weave. The weaver uses this file, once compiled, in order to perform the weaving at runtime, but no aspect unexpected at design time could be weaved once the application has started.

We think that an interesting dynamic-weaving issue is giving a system the ability to adapt to runtime-emerging aspects unpredicted at design time – e.g., a logging or monitoring aspect not considered previously to the application execution. A system that offers a development-time limited set of join points restricts this facility.

Another interesting issue is the ability to weave and unweave aspects at runtime, e.g., if you need a temporal execution metric you can weave an aspect that implements this metric, do the task, and as soon as it is finished, unweave the aspect making the application going back to its initial behaviour.

We have identified computational reflection [25] as the best technique to overcome the previously mentioned limitations, and applying it to a virtual machine we also achieve language and platform independence [26].

4 Reflection

Reflection is the capability of a computational system to reason about and act upon itself, adjusting itself to changing conditions [25].

We have identified [27] two main criteria to categorize reflective systems. These criteria are *when* reflection takes place and *what* can be reflected. If we take *what* can be reflected as a criterion, we can distinguish:

- *Introspection*: The system's structure can be accessed but not modified. If we take Java as an example, with its "`java.lang.reflect`" package, we can get information about classes, objects, methods and fields at runtime.
- *Structural Reflection*: The system's structure can be modified. An example of this kind of reflection is the addition of object's fields –attributes.
- *Computational (Behavioural) Reflection*: The system semantics (behaviour) can be modified. For instance, metaXa – formerly called MetaJava [28] – offers the programmer the ability to dynamically modify the method dispatching mechanism.

Taking *when* reflection takes place as the classification criterion, we have:

- *Compile-time Reflection:* The system customisation takes place at compile-time – e.g., OpenJava [29]. The two main benefits of this kind of systems are runtime performance and the ability to adapt their own language. Many static-weaving aspect-oriented tools use this technique.
- *Runtime Reflection:* The system may be adapted at runtime, once it has been created and run – e.g., metaXa. These systems have greater adaptability by paying performance penalties.

4.1 Meta-Objects Protocols

Most runtime reflective systems are based on Meta-Object Protocols (MOPs). A MOP specifies the implementation of a reflective object-model [30]. An application is developed by means of a programming language (base level). The application's meta-level is the implementation of the computational object model supported by the programming language at the interpreter computational environment. Therefore, a MOP specifies the way a base-level application may access its meta-level in order to adapt its behaviour at runtime.

As shown in Fig. 1, the implementation of different meta-objects can be used to override the system semantics. For example, in MetaXa [28], we can implement the class "`Trace`" inherited from the class "`MetaObject`" (offered by the language as part of the MOP) and override the "`eventMethodEnter`" method. Its instances are

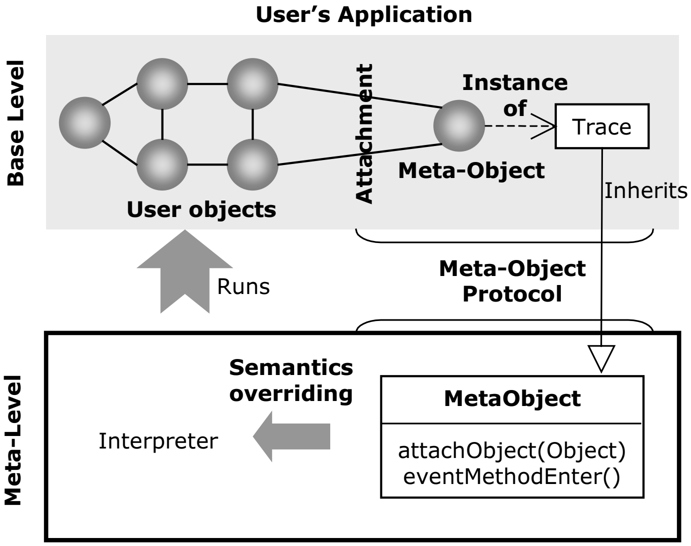


Fig. 1. MOP-based program adaptation.

meta-objects that can be attached to user objects by means of its inherited "`attachObject`" message. Every time a message is passed to these user objects, the "`eventMethodEnter`" method of its linked meta-objects will be called – showing a trace message and, therefore, customizing its message-passing semantics.

Some advanced dynamic-weaving AOP tools, like PROSE [7] or Handi-Wrap [31], use MOP-based reflective interpreters on its back-ends. We will apply this idea to the .NET platform virtual machine in order to develop a dynamic weaver independent of the selected platform and language.

5 Use of a Reflective Virtual Machine

An *abstract machine* is the specification of a computational processor not intended to be implemented in hardware. The term *virtual machine* is commonly used to denote a specific abstract machine implementation (sometimes called *emulator*, *interpreter* or *runtime*).

The use of specific abstract machines implemented by different virtual machines has brought many benefits to different computing systems. The most recognized are platform and language independence, application distribution over networks, application interoperability and direct support of specific paradigms.

First use of abstract machines obtained only one of these benefits. As an example, UNCOL was designed to be used as the universal intermediate code of every compiler, but it was never employed as a distributed platform [32]. Nevertheless, Java designers took many abstract machine benefits in order to build a platform for delivering and running portable, dynamic and secure object-oriented applications on networked computer systems. However, its design is focused on supporting the Java programming language, making it a difficult target for languages other than Java [33] – e.g. functional or logical applications.

Microsoft .NET platform is an abstract-machine based system whose common language infrastructure (CLI, ECMA-335) [13] has been designed from the ground up as a target for multiple languages, obtaining all the previously mentioned benefits. In this platform when any program written in any language is compiled, it is translated into *Microsoft Intermediate* Language, MSIL, which is processed by the VM, the *Common Language Runtime* (CLR). When the CLR is going to execute this code, in order to improve performance, it makes a *Just In Time* (JIT) compilation to specific native code. Thus, the application will not be executed by an interpreter.

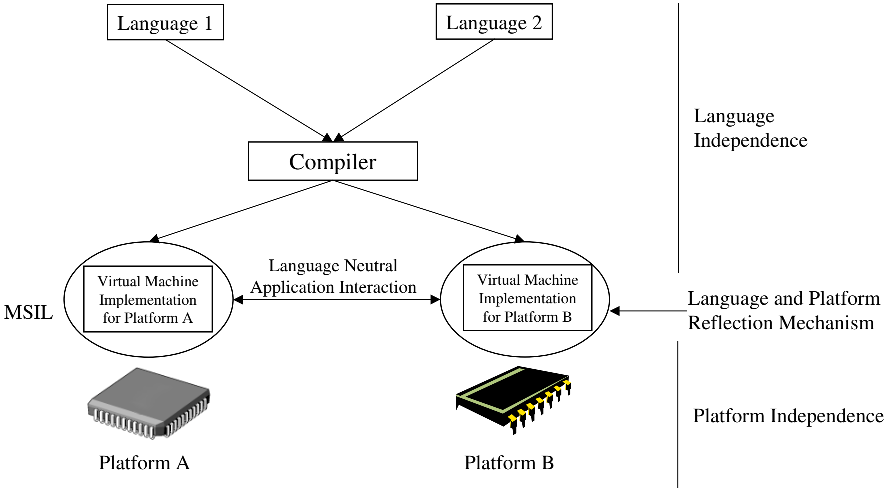

Language 1
Language 2
Compiler
Language Independence
MSIL
Virtual Machine Implementation for Platform A
Language Neutral Application Interaction
Virtual Machine Implementation for Platform B
Language and Platform Reflection Mechanism
Platform Independence
Platform A
Platform B

Fig. 2. Language and platform independent application adaptation.

As shown in Fig. 2 inserting reflection at MSIL level we obtain language independence, which is one of our goals. Whatever application written in any language is compiled with a CLR compatible compiler to MSIL code.

Different virtual machine implementations make the MSIL language platform neutral, because any application may be executed by different virtual machine implementations. Besides, at the MSIL level, any application could interact with another one, independently of the language they have been written in.

The virtual machine reflective mechanism offers application adaptation at runtime. This can be performed without the need of stopping and recompiling the running applications.

Last, an additional advantage of this scheme is that programs can be adapted without their source code. Microsoft .NET platform offers introspection facilities to obtain MSIL code from the binaries (bytecode).

6 System Design

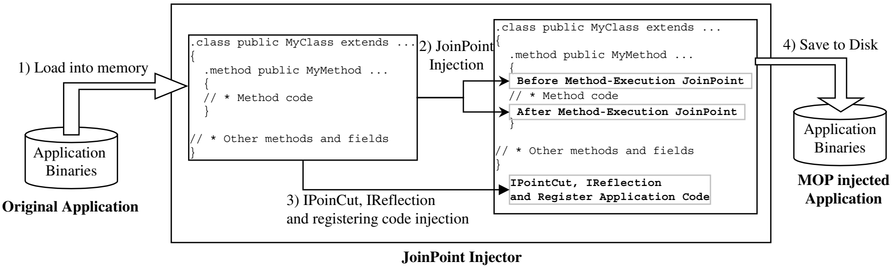
We are developing a system using reflection at the virtual machine computing environment, what allows adapting any application at runtime in a language independent way – both applications and aspects. Furthermore, those aspects that in our system do not need to be adapted at runtime can be woven statically obtaining a better performance.

We have studied different approaches to achieve our goals using the .NET platform:

1. The profiling API can be used as an alternative, employing the two interfaces `IcorProfilerCallback` and `IcorProfilerCallbackInfo`. Those interfaces work using an event-based mechanism, offering the ability to implement specific profiling tools.
   When the aspects system starts executing, it informs the execution engine about what events it wants to be monitored (supervised). At runtime, the execution engine would call the system through the `IcorProfilerCallback` whenever those events arise, and the system would inspect the executing code and adapt its behaviour if necessary.
   This approach has been used by CLAW [11] and presents some drawbacks. The system only can call *managed code* from the sole *method call* join point. This reduces very much the range of point-cuts offered by existing tools. Moreover, the profiler intercepts every user code as well as every Core Library method, making difficult the development of aspects and causing runtime performance penalties. Also, in case that the monitoring of another (previously unmonitored) event is demanded, there is no need to restart the application to inform the execution engine about the new requirements.
2. Another possible approach is to modify the virtual machine implementation offering a MOP reflective mechanism. This solution is quite complex and implies the use of a new modified abstract machine instead of the standard one. This tactic would imply a big drawback because we would loose portability.
3. Finally the selected alternative consist of a framework. Code must be injected in order to implement "join points" and "pointcuts", before any application comes into the system. A type of MOP is implemented, at the level of the .NET virtual machine, inserting calls to a server whenever a join point is reached. The server plays the role of a running applications registry. Aspects rely on the server to dynamically adapt the existing programs. By using this server, aspects may register (weave) to modify any application execution, and unregister (unweave) when this customisation is no more needed.

6.1 System Architecture

The system architecture is shown in Fig. 3 and Fig. 4. Before a compiled application is executed, it is processed and altered by our system. Our *join point injector* (JPI) inserts MOP-based computational reflection routines following the next steps:



```
.class public MyClass extends ...
{
  .method public MyMethod ...
  {
  // * Method code
  }

// * Other methods and fields
}
```

```
.class public MyClass extends ...
{
  .method public MyMethod ...
  {
  Before Method-Execution JoinPoint
  // * Method code
  After Method-Execution JoinPoint
  }

// * Other methods and fields
}
  IPointCut, IReflection
  and Register Application Code
```

Fig. 3. System architecture A.

1. The application is loaded into memory by means of the `System.Reflection` namespace. From the application binaries, this namespace is used to load the program MSIL representation into memory. Afterwards, the code will be manipulated as data.
2. Control routines injection by means of the `System.Reflection.Emit` namespace. Conditional routines to call future aspect's advice are injected in every join point (explained later). At first, this code does not invoke any routine, but modifying its condition later on its behaviour can be extended – as happens in most MOP systems
3. `IPointCut` and `IReflection` interfaces and application-registering code injection. `IPointCut` provides the interface to define different join points to be used in future adaptation. This functionality is the same as to AspectJ's *pointcut designators*.

   By using `IPointCut`, aspects may select a range of specific join points of an application to be called back each time the join point is reached (at method invocation, field access, or any of the existing join points explained later in this paper). This aim is achieved modifying conditions injected in the second point, assigning them true values and, therefore, causing aspect advice invocation whenever the join point is reached.

   `IReflection` interface allows other applications to access the application's information (intra-application introspection) – for security reasons, in .NET platform an application can only inspect itself; i.e. it cannot inspect another one. When aspects are being developed, it is commonly needed to access the structure of the application to be adapted: that is exactly the functionality offered by `IReflection`.

   This interface has been developed using mainly the `System.Reflection` namespace, allowing aspects to gain access to applications' information, invoking a method or even creating an object.

   Finally, necessary code to achieve application registration itself in the server at start up is injected – explained afterwards.
4. The modified application is saved to disk.

At start up the modified application will register itself, with a unique GUID, in the server (application registry) in order to allow aspects interact with it. When an aspect needs to adapt an application, the following steps are followed:

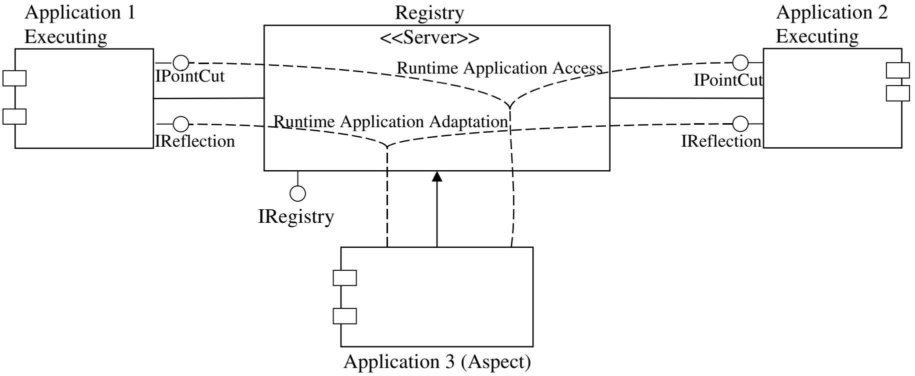


Fig. 4. System architecture B.

1. The aspect accesses the server and informs about the join points of the target application (the one to be adapted) in which it is interested in (in order to be notified when these join points are reached).
2. The server, by means of the `IPointCut` interface, requests the target application (the one to be adapted) to notify it when the specific join points are reached. What this interface does internally is to modify the appropriated conditions previously injected. So, when the join points are reached, the condition is satisfied and the server will be notified.

   With these first two steps, the aspect has designated the pointcuts.
3. When any of the requested join points are reached in the target application, the server is notified (by the target application) of this, and receives a reference to the application.
4. When the server receives a notification, it informs the aspects that have requested it (all of them), and pass them the reference. The server implements the *Observer* design pattern [34] to achieve this behaviour.
5. By means of this reference and the `IReflection` interface, implemented by the target application, the aspect can access the target application, obtain its structure, invoke target application's code, or run its own routines.
6. If an aspect is no longer needed, it informs the server in order to not been notified any more – when the join point would be reached in the target application.
7. If the server detects that there are no aspects left to be notified in the occurrence of a join point, it informs the target application of this, and, using the `IPointCut` interface, it modifies necessary conditions to false values. So, when the join points are reached, the server is not notified. This would obtain better runtime performance.

As we do all our work in code written in intermediate language, MSIL, we obtain two main benefits: language independence and platform independence in both the application to be adapted and the aspects that adapt it. In addition, it is not necessary at design time to know if the application is going to be adapted or not.

With our system, it is also possible to unweave (delete) an aspect when it is no longer necessary. A simple example is execution time metrics: 1) when we need to obtain something in a running application we will add an aspect to do this, 2) once the aspect has adapted the application we can obtain our metrics, and 3) when the metric is not needed we can unweave the aspect so the application returns to initial state.

Another important benefit we achieve is that we do not need the source code of an application to modify it as all the process is done from binary code that is translated to MSIL.

Furthermore, we do not modify the abstract machine so we can use our system whit any standard application.

6.2 Join Point Model

Our system offers a rich set of join points in order to achieve a power and flexibility similar to static systems. The existing joint points are:

- Method call
- Method execution
- Constructor call
- Constructor execution
- Static initializer execution
- Object initialization
- Field reference (rvalue)
- Field set (lvalue)
- Handler execution
- Property reference (rvalue)
- Property set (lvalue)

The last two join points are .NET specific. With these join points any aspect can manage completely the execution of an application giving the ability to implement any general purpose aspect.

6.3 Security

Security is a very important issue in our system's design because the system allows applications to inspect and modify the behaviour of other applications running in the virtual machine. The .NET common language runtime provides code access security, which allows the administrator to specify the privileges that each managed code assembly has, based on the degree of trust, if any, in that assembly. Following the same criterion, an aspect will not be able to weave an application that is not permitted in the system's "`security.config`" XML file.

At first, we have focused on developing a non-restrictive aspect weaver in order to allow aspects modify any application. Then we will use the .NET platform to build a secure aspect weaver. In order to accomplish this, we will create a new Security Class called "Weave" into the MS.NET "`System.Security.Permission`" namespace that inherits from the "`CodeAccessPermission`" class. By default, every application will not be granted this permission, meaning that applications could not be weaved. If we want an application to be weaved, we just have to specify it on its own XML configuration file, and the MS.NET framework will do the rest.

7 Performance

The main disadvantage of dynamic weaving is runtime performance [10]. The process of adapting an application at runtime, as well as the use of reflection, induces a certain overhead at the execution of an application [7].

Although there are aspects that will benefit from the use of dynamic weaving, this is not needed in many cases. If this situation occurs, static weaving should be used in order to avoid performance penalties. In our system, the weaving process could also be done statically; in this case performance penalty would be the same as other static weavers.

While developing aspect-oriented applications, the dynamic adaptation mechanism is preferable because it facilitates incremental weaving and makes application debugging easier in a "fix-and-continue" fashion [35], [27]. Upon deployment, aspects that do not need to be adapted at runtime should be woven statically for performance reasons.

Another performance limitation of our reflective platform is caused by the execution over a virtual machine. Nowadays, many virtual-machine-based languages are commercially used – e.g., Java [36], Python [37] or C# [38] – due to optimisation techniques such as just-in-time (JIT) compilation or adaptable native-code generation [39]. That has been the reason why Microsoft has launched its .NET platform based on this scheme.

8 System Benefits

The AOSD system presented on this paper injects code to applications obtaining MOP-based reflection at the virtual machine level, offering the following benefits:

1. Language independence. As we do all the process over intermediate language, MSIL, both the application that can be adapted and the aspects that can adapt an application can be written in any language, as the weaving process is done only with the code translated into MSIL.
2. Aspects and core functionality follow full independent lifecycles. Thanks to this we obtain a greater reusability of both, and it is easier to debug a program because the code is not tangled.
3. Platform independence. By doing all the process at a virtual machine, complete independence is obtained with respect to the platform over which is implemented.
4. We do not need source code to adapt an application, since we work only with MSIL. This is very useful if we have to modify code from third parties.
5. Use of a standard virtual machine. Our system does not need to modify the virtual machine, so it complies ECMA standard. Therefore, we can introduce any CLR standard application into our system.
6. The system offers a rich set of join points. The mechanism identified in our system does not limit the range of join points to capture in any application. Therefore, aspects can be developed the same way as in AspectJ, but weaving at runtime.
7. Finally, it is possible to adapt an aspect by means of another aspect. Aspects are common applications that work in our system, so they follow the same steps than other applications and can be adapted in the same way.

9 Conclusions

The principle of separation of concerns separates main application algorithms from special purpose concerns (typically orthogonal to the main functionality), building final applications by means of their main functional code plus their specific problem-domain concerns. The main benefits of SoC are higher level of abstraction, easier to understand, concern reuse, and increase of development productivity. AOSD is one approach to apply this principle. AOSD is a development methodology that provides explicit language support for modularising application concerns that crosscut the application functional code. By separating the application functional code from its crosscutting aspects, the application source code would not be tangled, being easy to debug, maintain and modify.

Most existing AOSD tools are static: once the final application is generated (woven), it will not be able to adapt its concerns (aspects) at runtime. This lack of adaptability at runtime has some limitations because there could be certain cases in which the adaptation of application concerns should be done dynamically, in response to changes in the runtime environment. In addition, some concerns may arise once the application is completely developed and running (and maybe it cannot be stopped).

Some existing dynamic AOSD tools have appeared, but they restrict the way aspects can adapt applications at runtime, and some of them are not really dynamic. Some modify the platform they run over, adding an important drawback: the need of executing the application over a modified platform instead of over a standard one.

Another important drawback that most of the existing tools present (both static and dynamic) is that they are language dependent, limiting their use out of its own programming language.

We identify reflection as a programming language technique that achieves dynamic application adaptability. It can be used to achieve aspect adaptation at runtime. Using computational reflection in a virtual machine environment, we obtain program adaptation at runtime regardless the language and platform. We are applying this idea to the new language independent Microsoft .NET platform.

The system is constructed by a joinpoint injector (JPI), which inserts code to any application simulating a virtual machine MetaObject Protocol. At start up applications register themselves in the server. When an aspect needs to adapt an application behaviour, it informs the server about the join points to be called back. Application events are captured by the server that is in charge of notifying interested aspects. Those may access applications to adapt them at runtime by means of the previously injected MOP.

As a result we have obtained a really dynamic aspect weaver (no need of predefined aspects nor source code), which has the same rich join point set as static weaving tools, is platform and language independent, and does not need to modify the ECMA standard specification.

References

1. Parnas, D., On the Criteria to be Used in Decomposing Systems into Modules. Communications of the ACM, Vol. 15, No. 12. (1972)
2. Hürsch, W.L., Lopes, C.V. Separation of Concerns, Technical Report UN-CCS-95-03, Northeastern University, Boston, USA. (1995)

3. Tarr, P., Ossher, H., Harrison, W., Sutton, S. N Degrees of separation: Multi-Dimensional Separation of Concerns. In: Proceedings of the 1999 International Conference on Software Engineering. (1999)
4. L. Bergmans. "Composing Concurrent Objects: Applying Composition Filters for the Development and Reuse of Concurrent Object-Oriented Programs". Ph. D. Dissertation. University of Twente. (1994).
5. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.M., Irwin, J. Aspect Oriented Programming. In: Proceedings of European Conference on Object-Oriented Programming (ECOOP), vol. 1241 of Lecture Notes in Computer Science, Springer Verlag.
6. Matthijs, F., Joosen, W., Vanhaute, B., Robben, B., Verbaten, P., 1997. Aspects should not die. In: European Conference on Object-Oriented Programming *(*ECOOP) Workshop on Aspect-Oriented Programming. (1997)
7. Popovici, A., Gross, Th., Alonso, G. Dynamic Homogenous AOP with PROSE, Technical Report, Department of Computer Science, ETH Zürich, Switzerland. (2001)
8. Pinto, M., Amor, M., Fuentes, L., Troya, J.M. Run-Time Coordination of Components: Design Patterns vs. Component & Aspect based Platforms. In: European Conference on Object-Oriented Programming *(*ECOOP) Workshop on Advanced Separation of Concerns. (2001)
9. R. Pawlack, L. Seinturier, L. Duchien, G. Florin. Jac: A flexible and efficient framework for aop in java. In Reflection'01*,* September (2001).
10. Böllert, K. On Weaving Aspects. In: European Conference on Object-Oriented Programming (ECOOP) Workshop on Aspect Oriented Programming. (1999)
11. John Lam. CLAW, Cross-Language Load-Time Aspect Weaving on Microsoft's Common Language Runtime. AOSD 2002 conference http://www.iunknown.com (2002)
12. Ortin, F., and Cueva, J. M. Building a Completely Adaptable Reflective System. European Conference on Object Oriented Programming ECOOP'2001. Workshop on Adaptive Object-Models and Metamodeling Techniques, Budapest, Hungary, June (2001).
13. ECMA. Standard ECMA-335: Common language infrastructure (CLI). Available from http://www.ecma-international.org/publications/standards/ECMA-335.HTM
14. The Free Online Dictionary. "spaghetti code". www.foldoc.org. (1997).
15. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G. Getting Started with AspectJ, Communications of the ACM, October (2001).
16. AspectJ homepage. http://eclipse.org/aspectj
17. O'Brien, L. The First Aspect-Oriented Compiler, Software Development Magazine, September (2001).
18. Weave.NET: Language-Independent Aspect-Oriented Programming. Distributed System Group at Department of Computer Science, Trinity College Dublin.
http://www.dsg.cs.tcd.ie/index.php?category_id=194
19. Wolfgang Schult and Andreas Polze Aspect-Oriented Programming with C# and .NET. in Proceedings of International Symposium on Object-oriented Real-time distributed Computing (ISORC) 2002, pp. 241-248, Crystal City, VA, USA, April 29 - May 1 (2002).
20. Wolfgang Schult and Andreas Polze Dynamic Aspect-Weaving with .NET. Workshop zur Beherrschung nicht-funktionaler Eigenschaften in Betriebssystemen und Verteilten Systemen, TU Berlin, Germany, 7-8 November (2002)
21. Wolfgang Schult, Andreas Polze Speed vs. Memory Usage - An Approach to Deal with Contrary Aspects. The Second AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS) at the International Conference on Aspect-Oriented Software Development, Boston, Massachusetts, March 17 - 21, (2003).
22. Douglas Dechow. "An Aspect-Oriented Infrastructure for a Typed, Stack-based, Intermediate Assembly Language" in the Doctoral Symposium at Object-Oriented Programming Languages, Systems and Applications (OOPSLA) (2002)
23. Zinky, J.A., Bakken, D.E., Schantz, R.E. Architectural Support for Quality of Service for CORBA Objects, Theory and Practice of Object Systems, 3 (1). (1997)

24. Marc Ségura-Devillechaise, Jean-Marc Menaud, Gilles Muller, Julia L. Lawall. Web cache prefetching as an aspect: towards a dynamic-weaving based solution. AOSD 2003 Proceedings, Pages: 110 – 119. (2003)
25. Maes, P. Computational Reflection. PhD. Thesis, Laboratory for Artificial Intelligence, Vrije Universiteit Brussel, Belgium. (1987)
26. Francisco Ortin and Juan Manuel Cueva. Non-Restrictive Computational Reflection. Elsevier Computer Software & Interfaces. Volume 25, Issue 3, Pages 241-251. June (2003).
27. Francisco Ortin, Juan Manuel Cueva. Implementing a real computational-environment jump in order to develop a runtime-adaptable reflective platform. ACM SIGPLAN Notices, Volume 37, Issue 8, August (2002)
28. Kleinöder, J., Golm, M. MetaJava: An Efficient Run-Time Meta Architecture for Java™. In: European Conference on Object-Oriented Programming (ECOOP) Workshop on Object Orientation in Operating Systems. (1996)
29. Chiba, S., Michiaki, T. A Yet Another java.lang.Class. In: European Conference on Object-Oriented Programming (ECOOP) Workshop on Reflective Object Oriented Programming and Systems. (1998)
30. Kiczales, G., Rivieres, J., Bobrow, D.G. The Art of Metaobject Protocol. MIT Press. (1992)
31. Jason Baker, Wilson Hsieh. Runtime aspect weaving through metaprogramming. AOSD 2002 Proceedings, Pages: 86 – 95 (2002)
32. Diehl, S., Hartel, P., and Sestoft, P. Abstract Machines for Programming Language Implementation. Elsevier Future Generation Computer Systems, Vol. 16 (7). (2000)
33. Meijer, K. and Gough, J. Technical Overview of the Common Language Runtime. Available online at http://docs.msdnaa.net/ark/Webfiles/whitepapers.htm. (2001)
34. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, (1995)
35. M. Dmitriev. Application of the HotSwap Technology to Advanced Profiling. In ECOOP 2002 International Conference. (2002)
36. Gosling, J., Joy, B., Steele, G. The Java Language Specification, Addison-Wesley. (1996)
37. Rossum, G. Python Reference Manual. Fred L. Drake Jr. Editor, Relesase 2.1. (2001)
38. Archer, T. Inside C#, Microsoft Press. (2001)
39. Hölzle, U., Ungar, D. A Third-Generation SELF Implementation: Reconciling Responsiveness with Performance. In: Proceedings of the Object-Oriented Programming Languages, Systems and Applications (OOPSLA) Conference. (1994)

Author Index