Lecture Notes in Computer Science 3511

*Commenced Publication in 1973*
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

Uffe Kock Wiil (Ed.)

# Metainformatics

International Symposium, MIS 2004
Salzburg, Austria, September 15-18, 2004
Revised Selected Papers

Springer

Volume Editor

Uffe Kock Wiil
University of Southern Denmark
Mærsk Mc-Kinney Møller Institute
Campusvej 55, 5230 Odense M, Denmark
E-mail: ukwiil@mip.sdu.dk

Library of Congress Control Number: 2005928378

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-10 3-540-27328-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-27328-8 Springer Berlin Heidelberg New York



Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11518358 06/3142 5 4 3 2 1 0

# Preface

This volume contains the final proceedings of the 2004 Metainformatics Symposium (MIS 2004). The event was held during 15–18 September 2004 in Salzburg, Austria at Salzburg Research.

MIS is an annual event focusing on finding common ground shared by researchers and practitioners in many different computer science areas who may use similar methods to achieve different ends. The goal is to find useful abstractions, notations, analytical frameworks, formalisms, and systems that improve our understanding of the underlying structures of various disciplines and families of systems within computer science. Ideally these constructs should have usefulness in conveying knowledge and understanding across disciplinary boundaries.

The proceedings of previous MIS events were also published by Springer in the Lecture Notes in Computer Science series: LNCS 3002 (2003), LNCS 2641 (2002), LNCS 2266 (2001), and LNCS 1903 (2000).

As with previous events in the MIS series, MIS 2004 attracted quality papers and brought together researchers from many different fields within computer science. We experienced interesting presentations and lively discussions in Salzburg. I hope that you will find the papers contained in this volume as interesting as the other members of the Program Committee and I have.

This volume would not have been possible without the help and assistance of many people. In particular, I would like to acknowledge the assistance of the Springer editors, Anna Kramer and Christine Günther, and the Executive Editor of the LNCS series, Alfred Hofmann.

March 2005 Uffe Kock Wiil

# Organization

## Organizing Committee

Siegfried Reich (Salzburgh Research, Austria)
Uffe K. Wiil (University of Southern Denmark, Odense, Denmark)
Peter J. Nürnberg (Aalborg University Esbjerg, Denmark)
David L. Hicks (Aalborg University Esbjerg, Denmark)

## Program Committee

Chair: Uffe K. Wiil (University of Southern Denmark, Odense, Denmark)

Members: Kenneth M. Anderson (University of Colorado, Boulder, USA)
Niels Olof Bouvin (University of Aarhus, Denmark)
David L. Hicks (Aalborg University Esbjerg, Denmark)
Peter King (University of Manitoba, Winnipeg, Canada)
David E. Millard (University of Southampton, UK)
Peter J. Nürnberg (Aalborg University Esbjerg, Denmark)
Siegfried Reich (Salzburgh Research, Austria)
Jessica Rubart (Fraunhofer IPSI, Darmstadt, Germany)
Manolis Tzagarakis (University of Patras, Greece)
Klaus Tochtermann (Know-Center, Graz, Austria)
Weigang Wang (University of Manchester, UK)

## Sponsoring Institutions

Salzburg Research, Austria
Aalborg University Esbjerg, Denmark
University of Southern Denmark, Odense, Denmark

# Table of Contents

## Computer Aided Composition

# Supporting Tools for Designing-By-Contract in Component-Based Applications

Antonio Coronato[1], Antonio d'Acierno[2], Diego D'Ambrosio[3], and Giuseppe De Pietro[3]

[1] DRR-CNR, Via Castellino 111, 80131 Napoli, Italy
coronato.a@ na.drr.cnr.it
[2] ISA-CNR, Via Roma, 52 Avellino, Italy
dacierno.a@isa.cnr.it
[3] ICAR-CNR, Via Castellino 111, 80131 Napoli, Italy
depietro.g@cps.na.cnr.it

**Abstract.** This paper deals with the modeling and the automatic implementation of constraints in component based applications. Constraints have been assuming an ever more relevant role in modeling distributed systems as long as business rules implementation, design-by-contract practice, and fault-tolerance requirements are concerned. Nevertheless, component developers are not sufficiently supported by existing tools to model and implement such features. In this paper, we propose a set of tools that enable developers both to model component constraints and to automatically generate component skeletons that already implement such constraints.

## 1 Introduction

In the last decade the nature of software systems has remarkably changed. In particular, the widespread diffusion of computer networks, as well as the constant growth of information systems responsibilities has led to new software architectures, no more single, huge, and strongly centralized systems, but highly-distributed ones.

Software distribution has initially been realized by decomposing software systems in modules, which implemented specific functions. They were able to run on distinct computers and adopted the remote procedure call paradigm as basis communication mechanism. More recently, software distribution has taken place by implementing and releasing cooperating software artifacts called components. A software component is a software element that can independently be deployed and composed without modification to support enterprise business processes [1].

A key point for the success of component-based software development practice has just been the possibility of having a more efficient and a relative easier software reuse. The wide diffusion of component-based architectures as a distributed system paradigm has also been pushed up by the emerging of middleware technologies like CORBA, DCOM, and more recently Web Services, which offer several facilities for developing and deploying cooperating distributed objects over heterogeneus platforms.

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 1 – 13, 2004.

In this work we focus on CORBA platforms. In particular, we are interested in developing new CORBA components applying a design-by-contract technique.

The design-by-contract method, which was primarily proposed by Bertrand Meyer in [13], is now a well established tool for developing reliable software systems. Such a technique fundamentally relies on the simple and compelling idea of designing systems as a set of cooperating abstract boxes that achieve their common goal by verifying specified contracts. A contract establishes what each participant of a collaboration has to do in order to get the promised results [6]. Contracts are implemented just as a set of constraints[1]. Design-by-contract has recently been adopted by Cheesman and Daniels in their component-oriented modeling process [6]. Such a process specifies workflows and activities that a designer has to perform in order to get complete component specifications. A complete component specification includes i) the component interface; ii) the inter-components collaborations; and iii) a set of contracts (constraints) as pre-conditions, post-conditions, and invariants that apply to the system components.

To build new CORBA components, developers model component architecture via the Unified Modeling Language (UML) [4]. Then, they model components interfaces by using the standard Interface Definition Language (IDL) [11]. Next, such interfaces are compiled by idl compilers to obtain component skeletons. Component skeletons are "empty" components that already integrate inter-component communication mechanisms, but do not have business logic. Finally, developers fill the component skeletons by adding the component business logic. It is worth to note that UML enables designers to produce low-fidelity models to capture high-level system characteristics in the early design phase, as well as high-fidelity models to specify low-level system details in the late design phase. The fidelity is to be intended as the measure of the correspondence between the model and the final system [12]. However, several component characteristics, like relationships and constraints, which can be modeled via UML, are not automatically implemented into component skeletons. This happens because component skeletons are automatically generated by idl compilers, which process idl models, but IDL doesn't provide support for modeling such characteristics. Indeed, IDL was devised to model only interfaces. It doesn't have to deal with structural features. As a consequence, component skeletons do not keep any track of several structural characteristics although they have been specified in the UML models. We can conclude that the component skeletons fidelity is quite low, where we define the *component skeleton fidelity* as the measure of how close a component skeleton is to its final implementation.

The target of this research activity is to provide tools able to automatically produce skeletons that already implement structural characteristics as invariants, pre-conditions, post-conditions, and guard conditions. This assures a higher degree of component skeleton fidelity. Moreover, this makes the implementation phase less time consuming and reduces development costs. In addition, it also reduces software faults by having reduced the manual programming impact.

[1] In this paper, the words constraint and contract are used as synonymous. However, it's worth noting that the term contract has a narrow semantic with respect to the term constraint.

In this paper, in section 2 we briefly report some motivations and related work. In section 3 we describe the proposed tools. In particular, we present i) the Constraint Description Language (CDL), which is a modeling language derived from the standard OCL [3]; ii) the Component Constraint Generator (CoCoGen), which is a tool able to process textual CDL constraint models and to automatically implement such constraints in enhanced component skeletons; and iii) the Component Constraint Modeler (CoCoMod), which is a visual UML-based modeling tool for specifying component interfaces and constraints; it is also able to automatically generate both IDL and CDL models. Sections 4 deals with a case study and presents the developing process. Section 5 concludes the paper and reports some directions for future work.

## 2 Motivations and Related Work

Modeling and implementation of constraints is a need for designers and developers. We have already cited the design-by-contract technique and its application in the component development process proposed by Cheesman and Daniels. There are others developing processes that use contracts in component based applications. Catalysis is just a further example [7]. However, contracts are used also in other specific activities. Indeed, designers can effectively model enterprise business rules as constraints [2]. In [19], a process for implementing collaborations among distributed components is proposed; such a process relies on the use of constraints. In [18], a technique for isolating faults is presented; this technique requires the insertion of contracts in the source code. Faults are isolated into the bundle of executed software instructions grouped by the last verified contract, and the first violated one.

Several examples of programming languages and tools, which offer facilities to support constraints, are described in literature. Eiffel [5] and Turing [16] are two relevant, constraint-oriented, programming languages. In [14], the Annotation PreProcessor (APP) tool for C programs is presented. In such a case, invariant conditions, pre-conditions, and post-conditions are formalized as comment lines directly in the source code; then, such "comment lines" are processed by a pre-compiler tool that translates the specified constraints into source code. The same approach is followed in [20] for the Java programming language. However, an important limit for existing tools, in our opinion, is that constraints have to be specified directly in the source code, so that the specification of constraints can take place only in the late implementation phase.

Such considerations can make easily understandable the need of having methodologies, processes, and tools able to support designers and developers who are willing of taking care of software constraints and logic assertions, possibly already in the design phase and without concerning about the implementation details. Unfortunately, current CORBA platforms do not provide any support for that as clearly denounced in [15].

In this paper, we describe an integrated set of tools able to assist designers both in modeling contracts in CORBA components and to automatically generate the proper source code. However, we do not define any new programming language, but we

provide a visual UML-based environment for specifying component interfaces and constraints. Finally, it's worth to note that we have extended the typologies of constraints. In particular, our tools enable to specify guard conditions and some inter-component relationships as constraints. For all such constraints, facilities that automatically generate the equivalent source code are provided.

## 3 The Proposed Tools

The use of the tools in the proposed approach is shown in figure 1. Designers use CoCoMod for producing UML models of components that describes both interfaces and constraints. CoCoMod automatically generates textual IDL and CDL models. Next, from CoCoMod, designers can run the regular idl compiler and successively the CoCoGen tool that realizes enhanced component skeletons.

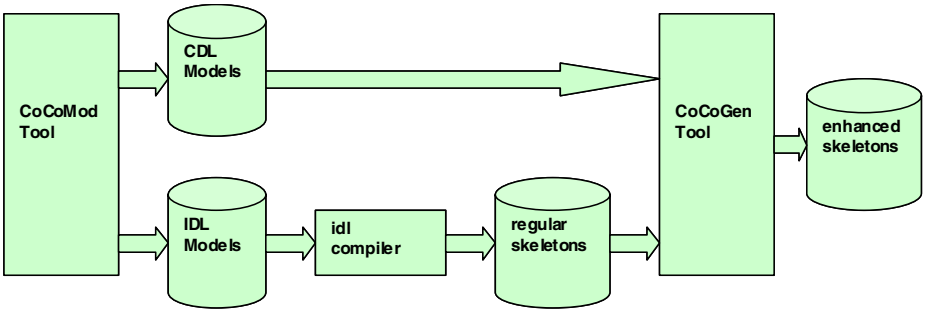



**Fig. 1.** High level architectural view

### The Constraint Description Language

The modeling language that we defined to formalize constraints is the Constraint Description Language (CDL). This is basically derived from the standard OCL and slightly adapted to component architecture features.

The original idea was to adopt the OCL as is but, on one hand, we experienced that only a subset of OCL expressions can automatically be implemented by generation tools. Indeed, because OCL is a declarative language, it is difficult to define generic implementation patterns for some OCL expressions. On the another hand, we have found useful to provide support for modeling guard conditions, which enables designers to specify some kinds of business rules, and for inter-components relationships, which are not supported by the standard IDL. The current version of OCL doesn't provide direct support for that, whereas the forthcoming version will take guard conditions into account as reported in the last Request for Proposal [21]. We have also found useful to have mechanisms for specifying state changes or

forcing the execution of operations once a condition is satisfied. This facility cannot be supported by OCL that is defined to be a side-effect free language.

These considerations have suggested the introduction of a new language, the CDL, which has been thought to be as close as possible to OCL; i.e. CDL syntax is almost the same as OCL. It also shares the same general properties, but CDL has also been devised to provide support for the automatic implementation of constraints in component based applications.

CDL (as well as OCL) is a formal language that remains easy to read and write because it doesn't rely on particularly complex mathematical constructs. It is not a programming language; therefore, it is not possible to write program logic. As OCL, it is a typed language, so that each expression has a type and, to be well formed, a CDL expression must conform to the type conformance rules of the language. Finally, as a specification language, all implementation issues are out of scope and cannot be expressed in CDL, but it enables designers to produce textual models, which can easily be processed by automatic tools to generate source code.

A brief description of CDL and OCL keywords is reported in table 1. The keywords supported by OCL and not supported by CDL indicates the characteristics that can not be automatically implemented by the current version of the tools described in the next sections.

It is worth to note that constraints are still specified via a declarative approach. This enables designers to model rules by abstracting away from any implementation choice. By this way, designers continue to work without the concern of implementation details.

**Table 1.** The CDL Language keywords

| Keyword | Description | OCL |
|---|---|---|
| *Context* | This keyword indicates the class or the operation to which the rule is referred to | ✓ |
| *INV* | This keyword indicates an invariant condition | ✓ |
| *PRE* | This keyword indicates a pre condition | ✓ |
| *POST* | This keyword indicates a post condition | ✓ |
| *GUARD* | This keyword indicates a guard waiting for a condition | |
| *ASSOCIATION* | This keyword indicates an association between two classes | |
| *TOWARD* | This keyword indicates the end point of an association | |
| *AGGREGATION* | This keyword indicates an aggregation between two classes | |
| *AGGREGATE* | This keyword indicates an aggregation between two classes. | |
| *AGGREGATEDTO* | This keyword indicates what is the aggregated object in an aggregation | |
| *COMPOSITION* | This keyword indicates a composition between two classes | |
| *COMPOSE* | This keyword indicates what is the composed object in a composition | |
| *COMPOSED* | This keyword indicates what are the composing objects in a composition | |
| *MULTIPLICITY* | This keyword indicates the multiplicity of a relation | |
| *SELF* | This keyword indicates the current instance of a class | ✓ |
| *EXECUTE* | This keyword indicates an operation to execute | |
| *@PRE* | This keyword indicates the previous value of a variable | ✓ |

## The Component Constraint Generator

CoCoGen is a generation tool that processes CDL textual models and then enhances the component skeletons generated by a regular IDL compiler. The tool is dependent from the target platform, i.e. a specific CORBA platform needs a specific constraint generator. Currently, CoCoGen supports two CORBA platforms: CORBA TAO [9], which is an open CORBA platform of the Washington University; and the Java CORBA ORB [8], which is a free CORBA compliant platform distributed with the J2SE environment.

CoCoGen architecture is shown in figure 2. It consists of the following main elements:

- *Coordinator* – This is the main thread that coordinates the processing phases;
- *Parser* – This module is in charge of analyzing CDL models and generating constraints data structures;
- *Processor* – This component implements constraints in the target component skeleton;
- *Scheduler* – This module coordinates the implementation of constraints, which needs proper nesting operations in the component skeletons.

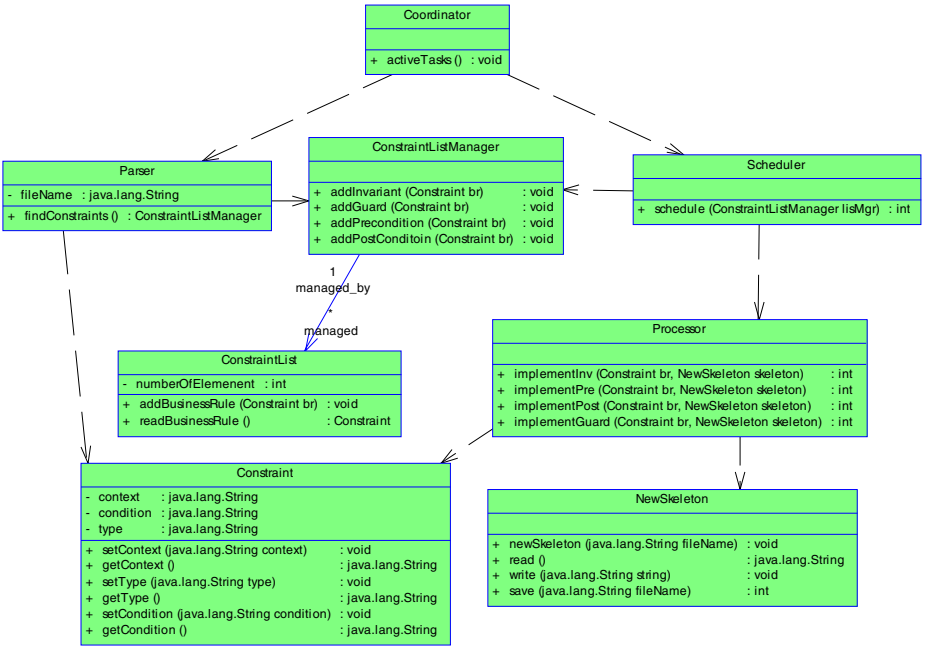



**Fig. 2.** CoCoGen architecture

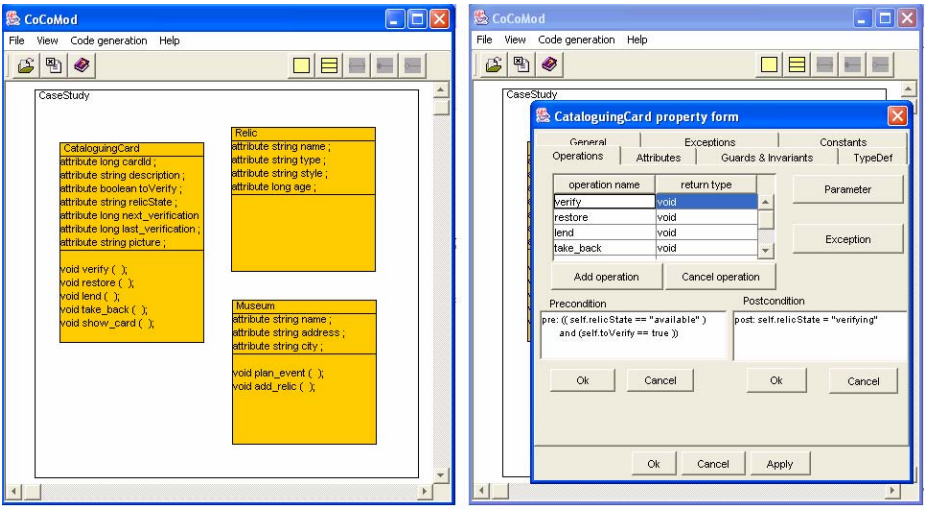
It is worth to note that the *Processor* is the only component which is platform specific. In other words, each CORBA platform requires a specific *Processor*.

CoCoGen operates in two phases. During the first phase, the *Parser* analyzes the input CDL models and then builds some data structures that are used during the second phase by the *Scheduler*. The *Scheduler* drives the *Implementer*, which executes specific implementation patterns depending on the type of the constraint.

Implementation patterns specify how to implement invariant, pre, post, and guard conditions in component-based applications. They have been defined in previous work [17].

### The Component Constraint Modeler

CoCoMod is a visual environment that enables designers to model both component interfaces and constraints. Interfaces are modeled as UML classes, whereas constraints are inserted as properties of the corresponding interface and expressed in CDL. From these graphical models, CoCoMod generates both IDL interface models and CDL constraint models. These textual models are platform independent. After that, from this tool, the developer can invoke the execution of a platform-specific IDL compiler. Two choices are currently available, tao_idl and idlj for the two distinct supported platforms (CORBA TAO and Java CORBA ORB). This step generates the regular component skeletons for the selected CORBA platform. Next, the designer invokes the execution of the CoCoGen tool that processes the CDL models so far generated, and then inserts in the regular component skeletons the proper source code for implementing constraints. The result consists of a set of enhanced component skeletons that already implement constraints.




**(a)** **(b)**

**Fig. 3.** CoCoMod functionalities

Figure 3 shows some functionalities of the CoCoMod tool. In particular, figure 3.a presents the workspace panel, from which designers can create project modules (packages) and interfaces. Figure 3.b shows the property panel for an interface. From this panel, it is possible to add not only interface operations and attributes, but also constraints as invariants, pre-conditions, post-conditions, and guards.

## 4 Case Study

This section presents a simple case study in order to show an application of the implemented tools. We focus on a system that must offer cataloguing and exposing functions for relics owned by an archeological museum.

Relics are classified by age, epoch, style, and type. A cataloguing card, which includes technical descriptions and pictures, is associated to each relic. Relics are also periodically verified in order to establish whether to restore them or not. Verifications, as well as restorations, are performed in specialized laboratories. Any time a relic is restored or verified, a verification timeout is set to establish how long to wait before having the next verification. Such a timeout must always range between maximum and minimum values (suppose 6 months and 24 months respectively) established for relics classes by generic criteria.

Museums can lend/borrow relics to/from other museums in order to organize special events. Relics can be lent only if they are available (not on verification/restoration). Likewise, relics can be verified/restored only if the verification timeout is expired and they have not been lent (they are currently available) to other museums.

In the following, the development activities are described.

Modeling the system

As shown in figure 3.a, three new components are designed: *Relic*, *Museum*, *CataloguingCard*. *Relic* is the component containing relic objects, which are owned by a museum. *CataloguingCard* stores information about the associated relic and its operations of verification, restoration, and lending.

A relic can be in the following state:

- Verifying – The relic is on verification in order to be restored
- Restoration – The relic is on restoration
- Lending – The relic has been lent
- Available – The relic is available in the museum

It has also been supposed to have a background procedure that periodically checks the verification timeouts and sets the toVerify attribute when the timeout has expired. The next_verification attribute of *CatalougingCard* realizes the verification timeout, which memorizes the number of months to wait for the next verification.

We can assume the following business rules and constraints:

- Business Rule 1 – Each relic must be periodically verified

- Constraint 1.1 – Before having the next verification, one must wait for at least 6 months and no more than 24 months.
- Constraint 1.2 – A verification can be performed only if the relic is available and its verification timeout has expired.

- Business Rule 2 – Relics can be lent to other museums.
  - Constraint 2.1 – A relic can be lent only if available

```
-- IDL model

module CaseStudy {
        struct Date {
                short day;
                short month;
                long year;
        };

        interface Relic
        {
                attribute string name;
                attribute string type;
                attribute string style;
                attribute long age;
        };

        interface CataloguingCard
        {
                attribute long cardID;
                attribute string description;
                attribute string picture;
                attribute boolean toVerify;
                attribute string relicState;
                attribute long next_verification;
                attribute Date last_verification;

                void verify( );
                void restore( );
                void lend( );
                void take_back( );
                void show_card( );
        };

        interface Museum
        {
                attribute string name;
                attribute string address;
                attribute string city;

                void plan_event( );
                void add_relic( );
        };
};
```

**Fig. 4.** IDL interfaces for the system components

```
-- CDL models

package CaseStudy ;

context CataloguingCard
inv: (( self.next_verification >= 6) and (self.next_verification <= 24 ))

context CataloguingCard
guard: if  ((self.toVerify == true )  and  ( self.relicState == “available” ))
         then
           execute self.verify()
         endif

context CataloguingCard::verify(): void
pre: (( self.relicState == “available” ) and ( self.toVerify == true ))

context CataloguingCard::verify(): void
post:  self.relicState = “verifying”

context CataloguingCard::restore(): void
pre: self.relicState == “available”

context CataloguingCard::restore(): void
post:  self.relicState = “restoring”

context CataloguingCard::lend(): void
pre: self.relicState == “available”

context CataloguingCard::lend(): void
post:  self.relicState = “lent”

context CataloguingCard::take_back(): void
post: self.relicState = “available”

endpackage;
```

**Fig. 5.** Constraint model

A couple of pre and post conditions are specified for the verify operation. In particular, the pre-condition verifies constraint 1.2, whereas the post-condition forces a new state for the relicState attribute. In order to implement all the previous business rules and constraints, further conditions have been formalized. Indeed, a guard condition, which executes the verify operation any time the executing condition gets true, has been inserted in order to implement business rule 1. Rule 2 affects the lend method, which can be executed only if the relic is available. Moreover, the lend operation must set the relic state to lending. Similar constraints must hold for restore and take_back operations. In addition, an invariant constraint has been set over the next_verification attribute in order to assure correct updates.

After having completed the modeling activity, CoCoMod generates the IDL model reported in figure 4 and the CDL model shown in figure 5.

From now on, to keep the example simple, we focus on the *CataloguingCard* component only.

Compiling IDL interfaces

This activity produces preliminary skeletons of components. We decided to implement components for the Java CORBA ORB platform. The idlj compiler is launched from the CoCoMod tool itself.

Generating components constraints

After having produced regular component skeletons, from CoCoMod we lauched the CoCoGen post-processor tool, which modified the initial component skeletons in order to implement constraints.

Figure 6 shows the resulting *CataloguingCard* component skeleton. Source code is added accordingly with specific development patterns. All software lines added to the original skeleton are shown as bold lines in the figure. However, to keep simple the figure, we reported only few pieces of the component skeleton.

```
package CaseStudy;

public abstract class _CataloguingCardImplBase extends org.omg.CORBA.portable.ObjectImpl
            implements CaseStudy.CataloguingCard, org.omg.CORBA.portable.InvokeHandler
{
                              .
                              .
 switch (__method.intValue ())
   {
                              .
                              .
                              .
     case 11:  // CaseStudy/CataloguingCard/_set_next_verification
     {
      int newNext_verification = in.read_long ();
      if ( (     newNext_verification >= 6)  &  (     newNext_verification <= 24 ) )
      {
           this.next_verification (newNext_verification);
           out = $rh.createReply();
      } else  throw  new org.omg.CORBA.UNKNOWN(  " operation impossible ");
      break;
     }
                              .
                              .
                              .
     case 14:  // CaseStudy/CataloguingCard/verify
     {
      if (( this.relicState ().equals ( "available") ) & (this.toVerify ()==true))
      {
           this.verify();
           this.relicState ( "verifying");
           out = $rh.createReply();
      } else  throw  new org.omg.CORBA.UNKNOWN(  " operation impossible ");
      break;
     }
```

**Fig. 6.** Enhanced component skeleton

Such a skeleton is now ready to be filled up with the business logic. This activity must be manually performed by programmers, who, however, do not have to care anymore about the implementation of constraints since it has been performed by the CoCoGen tool.

## 5 Conclusions

Constraints can no more be neglected while modeling and building component applications. We can also state that current components skeletons fidelity is quite low. The possibility of having automatic tools able to produce component skeletons with an higher degree of fidelity would have valuable effects in terms of development time, development costs, software correctness, and so on. One way to improve the component skeleton fidelity is given by automatically implementing component constraints.

In this paper we proposed and integrated environment composed by two tools, CoCoGen and CoCOMod. CoCoGen is a tool for generating constraints-related code into component skeletons; for such an aim, we formalized implementation patterns for invariants, pre-conditions, post conditions and guards. Our tool has been integrated in an UML compliant visual environment (CoCoMod) that we built to enable designers to visually define components interfaces and constraints, and to generate (for two major CORBA platforms) skeletons that already implement constraints.

Regarding our work in progress, some enhancements are being considered. First, we are extending CoCoMod with a graphic metaphor for modeling constraints, so that CDL will become an internal language and will no longer directly be used by software architects and developers. We are also considering other CORBA platforms together with the possibility of extending CoCoGen to other middleware platforms, such as RMI or SOAP.

## References

[1] G. T. Heineman and W. T. Councill, "*Component-Based Softwar Engineering*", Addison-Wesley publishing, 2001.
[2] R. G. Ross, "*The Business Rule Book*", Business Rule Solutions, 2nd Ed., 1997.
[3] J. Warmer and A. Kleppe, "*The Object Constraint Language: Precise Modeling with UML*", Addison-Wesley publishing, 1999.
[4] Hans-Erik Eriksson and Magnus Penker, "*UML toolkit*", Wiley publishing.
[5] B. Meyer, "*Object Oriented Construction*", Englewood Cliffs, NJ: Prentice-Hall, 1988.
[6] J. Cheesman and J. Daniels, "UML Components – A Simple Process for Specifying Component-Based Software", Addison-Wesley, 2003
[7] D.F. D'Souza and A.C. Wills, "Objects, Components, and Frameworks with UML: The Catalysis Approach", Addison-Wesley, 1999
[8] Sun Microsystems, "*CORBA Technology and the Java2 Platform, Standard Edition*", available at http://java.sun.com/j2se/1.4.2/docs/guide/corba/index.html
[9] http://www.cs.wustl.edu/~schmidt/TAO.html
[10] "*ORBIX 2000 Tutorial*", available at http://www.iona.com/docs.

[11] ISO/IEC 14750 standard.
[12] N. Medidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins, "*Modeling Software Architectures in the Unified Modeling Language*", ACM Transactions on Software Engineering and Methodology, Vol 1, N. 1, January 2002, Pages 2-57.
[13] B. Meyer, "*Applying Design by contract*", IEEE Computer, October 1992, Pages 40-51.
[14] D. S. Rosenblum, "*A Practical Approach to Programming with Assertion*", IEEE transactions on Software Engineering, Vol. 21. No. 1, January 1995, Pages 19-31.
[15] J. M. Jézéquel and B. Meyer, "*Design by contract: The Lessons of Ariane*", IEEE Computer, January 1997.
[16] R. C. Holt and J. R. Cordy, "*The Turing Programming Language*", Communication of ACM, Vol. 31, No. 12, December 1988.
[17] A. Coronato, M. Cinquegrani, and G. De Pietro, "*Adding Business Rules and Constraints in Component Based Applications*", in proc. of the 2002 International Symposium on Distributed Objects and Applications (DOA 2002), Irvine, California, USA - LNCS 2519.
[18] L. C. Briand, Y. Labiche, H. Sun, "*Investigating the use of analysis contracts to support fault isolation in object oriented code*" ACM SIGSOFT Software Engineering Notes , Proceedings of the international symposium on Software testing and analysis, Volume 27 Issue 4.
[19] E. Cariu, A. Beugnard, and J. M. Jezequel, "*An Architecture and a Process for Implementing Distributed Collaborations*", in the proc. of the 6th Int. Enterprise Distributed Object Computing conference (EDOC'02), IEEE Computer Society Press.
[20] R. Kramer, "*iContract – the Java Design by Contract tool*", in proc. of International Conference of Object Oriented Language and Systems (TOOLS 26, USA'98), IEEE Computer Society Press.
[21] OMG, "Response to the UML 2.0 OCL RfP (ad/2000-09-03)", Revised Submission, Version 1.6, January 6, 2003

# Access Rights – The Keys to Cooperative Work/Learning

Thorsten Hampel

University of Paderborn, Computer Science, Heinz Nixdorf Institute,
Fürstenallee 11, 33102 Paderborn, Germany
hampel@uni-paderborn.de

**Abstract.** It is almost impossible to distinguish and keep track of all the cooperative systems and computer-supported learning environments currently available. One of the key characteristics of any cooperative system is the chosen *access* rights model. Most systems of this sort employ a model that is specifically tailored to *one* application area, e.g. a number of fixed roles. This article begins by examining the different basic characteristics of access rights in the area of cooperative work, and then goes on to present an open and flexible rights model. In addition to covering classical access rights, this model enables rights to be delegated to others and inherited contextually. The presented model is implemented in the cooperative sTeam open-source work environment. Finally we will present some evaluation results of using sTeam at various courses at our university.

## 1 Introduction

Promising cooperative knowledge organization and modern computer-supported cooperative learning are characterized by open and flexible forms of interpersonal cooperation and handling of material. One feature of such self-organized and open environments is that they allow user groups to be created flexibly and on a self-organized basis. Another is the need to make the handling of material (documents) adaptable to this new and flexible form of cooperative work [12]. Access rights play a crucial role here. They govern the assignment of objects to groups of users or learners, thus clearly defining individual users and user groups. Such access structures must, however, be flexible enough to accommodate different cooperative processes between users, e.g. the exchange of documents, without the need to explicitly make complicated adjustments to the respective user rights.[1]

A key concept here is, for example, the context in which a document (object) is embedded.

The process of assigning user rights must therefore be seen as an essential criterion of a cooperative learning environment's self-administration and self-organization.

[1] In practice, many systems avoid the problem of having to make manual adjustments to a large number of access rights when handling objects – e.g. moving a document – by choosing quite open rights structures: "all users have full access rights". However, such an approach can only be considered a viable solution in limited use contexts.

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 14 – 31, 2004.

Powerful mechanisms for assigning, transferring (delegating) and deriving access rights from the use context are the basic elements of a learning environment that can be largely organized and structured by the learners themselves – in other words, a user-centred learning environment.

By the early 1990s, Ellis et al. had recognized that all existing CSCW system concepts were overly complex and said nearly nothing about mapping the underlying rights models to the user interface [5]. This conclusion must be qualified when applied to the situation today, but it continues to be valid: *existing access rights models of cooperative systems are either trivial or highly complex.*

There are in addition a large number of generic models and concepts for administering user rights in non-cooperative applications. Unfortunately, such models cannot be applied to cases in which cooperative teaching and learning or work environments are supported. This might be one of the reasons why in recent years few cooperative learning environments have been equipped with powerful user rights models [24, p. 51]. The rights systems of cooperative environments can also be extremely complex. Requirements with respect to clarity (comprehensibility) and suitability of the user interface thus also acquire importance.[2]

Elaborated models of access rights are found in many systems, such as [2], [3], [4], [9], [18] or [23].

The user rights model presented here is flexible and adaptable enough to cover the whole range of cooperative knowledge organization processes. And it is designed to minimize the complexity and effort involved in setting user rights. Key concepts here are the specific *right to delegate user rights, the inheritance of access rights and their derivation from the context/environment of an object.*

## 2 Access Rights – The Key to Cooperative Knowledge Organization

A general distinction should be made between authentication, i.e. logging on to a system and thus securing a specific status depending on the respective user ("Who are you?", "Who said that?"), and authorization, i.e. checking access rights between documents, objects and a person ("What are you allowed to do?", "Who is allowed to access this document?").[3] Conventionally, operating systems have made access rights to a specific file dependent on membership of the relevant user group. In the UNIX operating system, for example, the user's or user group's access rights to a file are defined in terms of only three attributes: *read, write, execute.* These rights are evaluated depending on a user's membership of specific groups.

The classical model for assigning access rights to users and user groups (domains) and objects is the Access Control List (ACL) as defined by [19], [20]. Lampson's notation consists of a set of objects $O$, a set of domains $S$ and an access matrix $A$.

[2] End users must be able both to easily recognize existing or refused access rights and to modify such rights (see [5]).

[3] Henceforth, the terms "access rights" and "access control" are used synonymously.

Objects are the system's objects/files, access to which is protected, e.g. the material in cooperative knowledge spaces. The definition of domain includes all entities, e.g. users or user groups that are assigned access rights to objects by the access function $f$ over the matrix $A$. Instead of "domain", Shen and Dewan use the term "subject" $S$, i.e. an entity that wishes to have access to an object [24, p. 52ff.]. Analogously, we use below a tripel *(S, O, A)* to describe the security status of the model.

The access matrix $A$ creates a relation between objects and subjects, i.e. users and groups. Each element of the matrix specifies via a set of access rights $R$ (often implemented by a bit string) which access types $r_1,..,r_n$ exist between $S$ and $O$, e.g. *read, write, delete*. These rights specify the access of a user or a group to the relevant object. In the literature, a combination of such access rights is called a "view".

The number of possible rights within a view varies considerably from one system to another. Some systems use a very large number of individual rights in order to be able to influence every aspect of cooperative work by a corresponding right (typically systems providing document-management functions). Other systems manage with only a few access rights, which are more general in nature (e.g. systems that primarily provide synchronous cooperation support).[4]

The concept of cooperative knowledge spaces[5] presented here and implemented in sTeam (see [12]). can be considered as belonging to the second group of systems: those with a small number of access rights. In a cooperative knowledge space, not every option (action) is controlled by its own access right. A large number of access rights can, for example, be subsumed under the basic rights *read, write (delete), execute, move, insert and annotate.*

If we now consider the columns of the matrix, we find specified for each object, in the form of a list, the subjects that have access to that object, and thus the users and user groups (or roles) that are allowed access to the object. Each object is thus assigned an Access Control List (ACL).

This enables a function $f$ to be easily defined and implemented, which yields for each user or user group u,g $\in S$ and each object o $\in O$, depending on the set of access rights $r_i \in R$, the values "allowed" or "denied":

$$\begin{aligned} f &: S \times O \times R \rightarrow \{allowed, denied\} \\ R &= \{read, write, delete,..\} \\ S &= \{User1, User2, User3,.., Group1, Group2,.., Role1, Role2,..\} \\ O &= \{obj1, obj2, obj3,..\} \end{aligned} \tag{1}$$

It can thus be determined for every access type $r_i \in R$ to what extent a user $u \in S$ or a group $g \in S$ has the access right $r_i \in R$ (e.g. *read*) to the object $o \in O$. In the above case, the access function $f$ checks whether there is an element $r_i$ for the user $u$ in the ACL of the object *o:*

[4] Of course, the latter, with their limited number and relatively close coupling of participants in a cooperative session, do not require such a complex system of rights as typical asynchronous systems.

[5] For the concept of cooperative knowledge spaces see [11] and [12].

$$f(u,o,r) = \begin{cases} allowed, & \text{if } u \in o.ACL \text{ for } r \\ denied, & \text{if } u \notin o.ACL \text{ for } r \end{cases} \quad (2)$$

The presented concept provides for each object the basic rights *read, write, delete* (i.e. write access), *move, execute, insert*[6] and annotate as well as the transfer/delegation of rights.

The delegation of rights is an extension of the administration right concept, as proposed by [22, p. 262].[7] This right allows the explicit modification of an ACL and is thus a first important step towards decentralized administration.

The right to transfer access rights enables cooperative structures to be implemented, in particular making material available to co-users. Here, either a document is made fully available to the cooperation partner (transfer of administration of the document) or merely the right to set access rights is given (a document remains in the possession of the user, an individual user or group of users merely being given the right to assign arbitrary access rights).

## 3 Inheriting Access Rights

In daily practice, authorizations and rights are a field that is both dynamic and at the same time characterized by social laws and norms. We consider it quite natural, then, that material can, when being passed from one person to another, adapt its status in a number of ways to the respective use context, the actors or the place. For instance, an exam paper changes its status from being a highly sensitive document, for the eyes of the lecturer only, before the examination to being a publicly available document that can be freely discussed and annotated after the examination. It would appear essential, particularly in cooperative teaching and learning processes, to get away from a rigid assignment of access rights to documents.

The dynamic delegation/derivation of rights falls into four distinct categories:

*I) derivation of access rights from the social group structure,*
*II) inheritance of access rights from (a) the environment or (b) a fixed reference object,*
*III) definition of a whole group of persons with administration rights, and*
*IV) transfer (delegation) of the right to create new rights, i.e. transfer of the responsibility for an object.*

---

[6] The insert right refers to the right to insert elements into the environment of an object, e.g. the insertion of material within a space/area. This is, initially, separated from the write-access right to the area to enable the modification of the area itself (e.g. changing of attributes) to be controlled.

[7] In the literature, the transfer (delegation) of access rights was first discussed in connection with authorization models for relational databases [8]. Authors like Bertino et al. take up these ideas, adding concepts such as negative rights and extended mechanisms for delegating and revoking the delegation of rights [1].

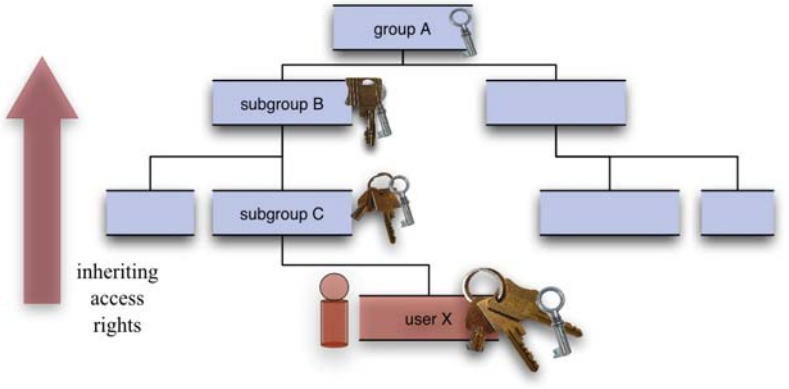



**Fig. 1.** Inheriting Access Rights from the Group Structure

### 3.1 I) Inheriting Rights from a Group Structure

Definitely the simplest way to obtain rights to access documents and material is from the user and user group structure. If rights are given to an entire user group, all members of that group including all members of subgroups acquire those rights (cf. Fig. 1). Such a mechanism can be used to map rights derived from users' social status, i.e. their permanent membership of a user group, but also to express dynamic processes such as role membership. For instance, if in the course of a discussion a participant is appointed to act as moderator and is admitted to a corresponding group, that participant automatically acquires the access rights to documents and material that are tied to the role of moderator or the access rights to the environment of the discussion process.

To this extent the rights model for cooperative knowledge spaces presented here is chosen to allow flexibility with respect to the assignment of persons to groups or roles. The sTeam system allows the flexible rights concept to be transferred to the management and administration of the group structure, enabling rights to groups to be administered on a decentralized basis and to be flexibly adapted. Access rights are delegated via the group hierarchy of parent and child groups.

Besides inheritance from the group structure, the context, i.e. the environment, of an object (document) is of particular importance for its accessibility (access rights).[8] The chosen concept allows us to specify for a particular object whether the right to access it is to be derived from the object's environment (contextual derivation of access rights) or from an explicitly specified object (semantic derivation of access rights).

[8] One need only think of various real-life metaphors: a document that is lying on a desk, open to all eyes, or is actually pinned on a notice-board has a quite different status – and thus different access rights – from a document that is kept locked in a safe or in someone's wallet.

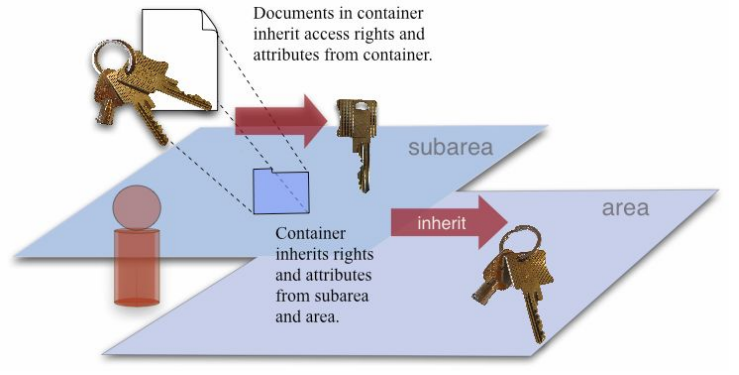



**Fig.2.** Inheriting Rights from the Environment

### 3.2 II a) Inheriting Rights from the Environment – Contextual Derivation of Access Rights

The process of inheriting rights from the environment allows us, for example, to specify rights to access containers or entire spaces/areas, which delegate the rights to the material they contain. This enables cooperation processes to be designed simply and flexibly, the material's structuring elements, i.e. environments, providing it with the necessary rights for a groupwork or learning process. In an approach of this sort, moving or transferring a document also causes the access rights to the document to be adapted to the new conditions. The context from which the rights are inherited may be either a container or a virtual space, or any other type of object (e.g. a user's rucksack).

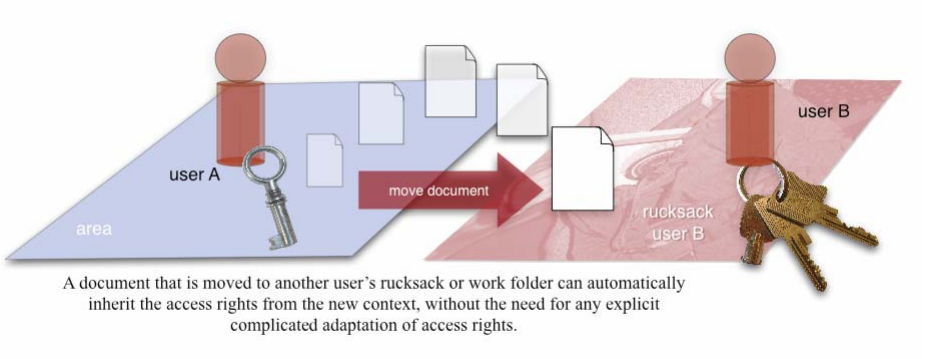



**Fig. 3.** Contextual Inheritance of Access Rights

The inheritance of rights is not confined to one step of the hierarchy; for example, starting from the inheritance of rights from the immediate local environment, the multiple inheritance of rights from arbitrary higher steps of the hierarchy is possible.

Such a concept is particularly useful for the contextual inheritance of rights from the respective environment: the object inheriting from the container, the container in turn inheriting from its area (cf. Fig. 2).

In addition, the contextual-inheritance mechanism can be used to map very neat cooperation models to the user-rights structure. [9]

For instance a document that is moved to another user's rucksack or work folder can automatically inherit the access rights from the respective object, without the need for any explicit complicated adaptation of access rights (cf. Fig. 3).

### 3.3 II b) Derivation of Access Rights from a Fixed Reference Object

When creating new documents or inserting documents, document-management systems typically select a default setting for access rights. Experience with various systems in everyday practice has shown that such a general setting of rights means that there is a continual need to globally adapt such rights. For this purpose, some systems provide the option of setting rights for a number of objects in one step, but this has to be explicitly applied after every modification to the overall structure.

Our experience shows that the need to continually adapt rights has a highly negative effect on the handling of material (e.g. a presentation/lecture) that contains a large number of files. Every instance of moving or transferring material involves explicitly adapting the rights for all files. By contrast, a rights *delegation mechanism* enables all access rights to be derived from a *central fixed object.*

The option of deriving rights from another specified object ties the rights, irrespective of the document's location, to the rights of a *fixed reference object* (inheritance object). This may be a space/area, a container, any other object or a user.

Deriving access rights is particularly important in order to simplify as far as possible the handling of material within spaces/areas and make it as smooth as possible for participants in a cooperation process. Ideally, there are in practice only very few rights directly tied to documents. Such an approach makes it much easier for learners to adapt the structure of teaching material flexibly to their own concrete needs. In addition, the creation of new objects can be considerably simplified by cleverly presetting the rights to new documents, e.g. by specifying a container or space/area as a rights inheritance object, so that rights do not have to be explicitly respecified or adapted.

### 3.4 Responsibility for an Object

The classical approach of assigning objects to users is based on a simple owner relationship. Many systems define the generator of a file as its "creator" or "owner", i.e. the person responsible for that file. As a direct consequence of this, an "owner" exercises the right to modify arbitrary access rights to the object in question. This can be illustrated, for example, by the UNIX operating system, which explicitly designates the owner of a file and also provides mechanisms for transferring the ownership

[9] See also [26].

of a file to another user. It is important to note that the owner need not automatically possess all the rights to the file's objects – but he can at all events assign himself such rights.

The administration of objects can be divided into three broad conceptual classes: *the option of performing administrative tasks on an object (e.g. setting rights), delegating administration rights to a whole group of persons, and delegating responsibility for selecting rights to other persons.* The last two ways of making administrative tasks more flexible are considered in greater detail below.

### 3.4.1 III) Trusted Group

The simplest way of performing administrative tasks on an object is something that can usually be done by the owner of the object. This approach can be generalized by tying not a *single* owner but a *whole group* of trustworthy persons/administrators (trusted group) to an object. The underlying concept is henceforth called the delegation of rights. Current research features various approaches that are broadly based on the delegation of rights.[10]

Approaches involving the transfer of documents from one worker to the next – approaches well known from workflow management – can be applied analogously to the concept of delegating rights. Besides explicitly transferring a document to another person, there are proposals to offer the option of revoking such a delegation step. The presented rights model would implement this approach by removing the person in question from the group of those authorized to administer the object.

The trusted group is not homogeneous in every case. It is, for example, still possible to identify the owner of the document within the trusted group. This person is still authorized to delegate to additional users the rights to this document.

Conceptually, the idea of being able to define trusted groups fits in perfectly with the concepts for cooperatively working on and arranging documents. Applied to the idea of the self-organization of learning groups, such a mechanism can be used to create whole groups of persons enabled to administer material.

### 3.4.2 IV) Right to Delegate Rights – Delegating Responsibility for an Object

The delegation of rights does not wholly solve the problem of decentralized administration or self-organized administration of material. Only delegation of the right to set new rights (a right that is typically exercised only by the owner/creator of a document), allows free structures to be created when handling material. It may, for instance, in

[10] Stiemerling and Wulf, for example, propose a concept that extends user rights to include the attributes of "trustworthy persons" [25]. It is a type of protocol for extending a trusted group. Though the basic idea was developed in the context of groupware, it can be applied perfectly well to teaching- and learning-support scenarios. If a person attempts to open a document without having the required read rights (or to write a document without having the necessary write access), a notification mechanism is used to contact a number of previously specified persons to ask how far this right can be assigned. Various strategies are conceivable here – e.g. a set term for raising objections. Such a concept allows greater freedom when subsequently working on material because rights can be dynamically adapted and extended by the use of documents.

cooperative processes be necessary to transfer material entirely to another user or to delegate the administration of this object.

An interesting aspect, and certainly an important step towards the decentralized handling of documents, is – besides delegating the right to administer access rights – the delegation of responsibility for a document. Normally, the creator of a document is also its owner. Various approaches are conceivable for transferring the ownership of a document. Most of them depend primarily on skilful implementation in an appropriate user interface.

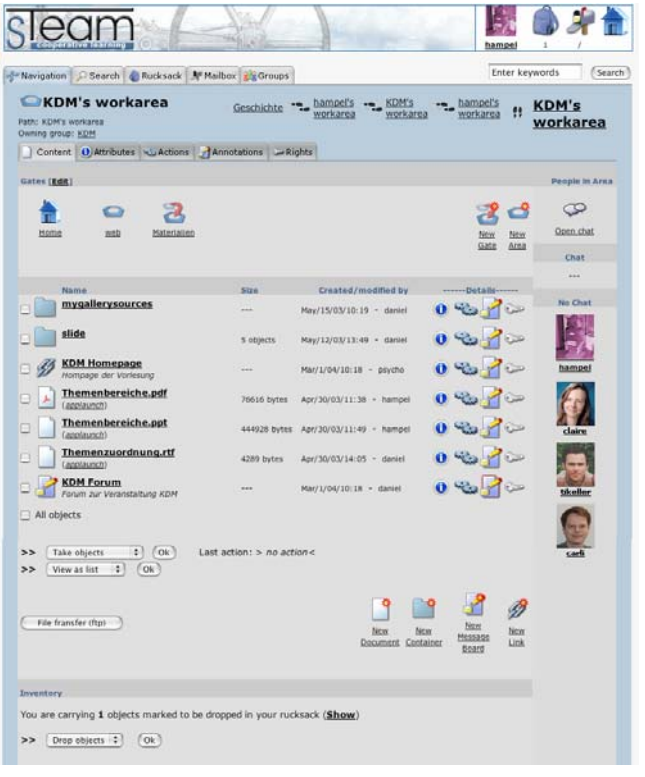



**Fig. 4.** The Paderborn Open Source sTeam system – structuring information in Teams

The model proposed in this article introduces a special right to transfer (delegate) access rights, the so-called *"sanction right"*. An *ownership transfer right* is thus introduced for each object. This right can be assigned to individuals or groups of people. The possessors of such a right are enabled to transfer ownership of the document to others, and thus extend the right to administer rights. Our experience shows that the ability to selectively control the delegation of rights in this way is one of the key characteristics of a cooperative work environment that is self-organized and administered on a decentralized basis.

### 3.5 Authorizations Outside the Model – Group of Administrators

Besides careful user and user group structuring, it would appear essential in practice to introduce a group of users that are able to generally access any object within the environment without the explicit setting of rights in ACLs. This is essential for administrative reasons – considering the possibility that users could, by mistake, revoke authorization for them to access their own files – even though such a general assignment of rights is by no means unproblematic from the point of view of data protection.

In the sTeam concept, an approach featuring specific group rights is chosen, enabling rights to all objects within the system to be specified for each user group. This is a completely opposite approach to specifying rights to documents for users and groups. Such group rights allow, for example, a group of administrators to be granted the right to modify rights to all documents within the system. At the same time, different types of cooperation rules can be defined. For instance, it can be defined for a particular group that the right to read documents is available to every group member. Specifically, this involves defining a series of rights for user groups that can have fundamental effects on the composition of cooperative knowledge spaces (e.g. read rights for all documents in a group space).

### 3.6 Rights Groups

Motivated by a large number of "collaboration rights" [24], i.e. a large number of individual rights such as can occur in a cooperative system, Shen and Dewan introduce the concept of rights groups [ibid., p. 53ff.]. Here, a transitive contained-in relation is defined for individual access rights. This enables complex authorizations to be reduced to elementary access rights. The model proposed here contains no restrictions with respect to the number of elementary access rights, but to ensure clarity and comprehensibility for users, their number should be kept to, say, eight at the most.

More complex access rights are implemented by special applications or clients, e.g. the right to make entries in a cooperative groupware calendar can be implemented by the respective client to the write or insert right for a groupware-calendar object. In this way, most complex authorizations can be reduced to write or execute rights for specific objects. These can be represented specifically by the user interfaces of the cooperative applications. The kernel system does not necessarily need a new, separately administered right.

### 3.7 Roles – User-Tied Authorizations

Early on in their work on the cooperative editor CES and the groupware calendar PCAL, Greif and Sarin discuss the use of roles as a way of abstracting different access rights, decoupled from the access rights models of the underlying operating system [7, p. 199ff.]. Roles are characterized by rights to access or modify

documents and objects. Analogously, Ellis et al. define roles as privileges, responsibilities and attributes assigned to a person.[11]

Many content-management – and also hypertext – systems use roles as a pure abstraction for the definition of a combination of access rights [6]. The role concept is, however, even considered useful in business-application contexts. To implement the roles in cooperative knowledge spaces, associations between access rights and roles as well as between roles and users have to be administered. Applied to the developed model of user groups and access rights, roles can be implemented by the *flexible creation of specific user groups or the flexible assignment* of users to user groups. Group rights assigned to a specific user group conform to generally assigned role rights. Rights to access specific material that are assigned to a role are implemented by rights of the corresponding user group to access the objects. Roles can be flexibly assigned by adding a person to a user group or revoked by removing a person from a user group. The main effort involved in providing roles is for their visualization in the respective user interface and for continuously matching the chosen roles to the structure of the learning process.

## 4 Our Experiences – Evaluation of the Presented Model

Besides testing and evaluating existing tools and system solutions, for some years now we have been designing and implementing architectures and tools for cooperative knowledge organization. The $^{open}$sTeam open-source environment (cf. Fig. 4) is a product of these development efforts [10], [12]. $^{open}$sTeam is not so much a concrete learning environment or a learning system based on specific didactic principles as an open-source environment for cooperative knowledge construction that can be fleshed out according to the respective user's needs and wishes.

In addition to the design of architectures for cooperative knowledge construction, the environment features a theoretical approach and a number of metaphors. These allow us to differentiate between technical problems on the one side and didactic and pedagogical problems on the other, and thus enable us to derive the essential technical issues relating to cooperative knowledge construction. The result is the concept of media functions [11], [12], which allows elementary cooperative actions in handling material to be viewed as part of a knowledge construction process.

Parallel to the design of various theoretical models and the construction of environments supporting cooperation, the developed architecture and its concrete implementations in the form of the sTeam system and its web-based interfaces are subject to constant testing in everyday practice [13], [14]. The feedback obtained is continuously incorporated into the design processes for the above-mentioned infrastructures and used to further refine the underlying theoretical concept.

Therefore the presented model of access rights is the result of a numerous evaluation of the sTeam-cooperative work/learning system in daily use. Different access rights models have been evaluated in a number of contexts.

---

[11] "A role is a set of privileges and responsibilities attributed to a person [...]. Roles can be formally or informally attributed." [5, p. 46].

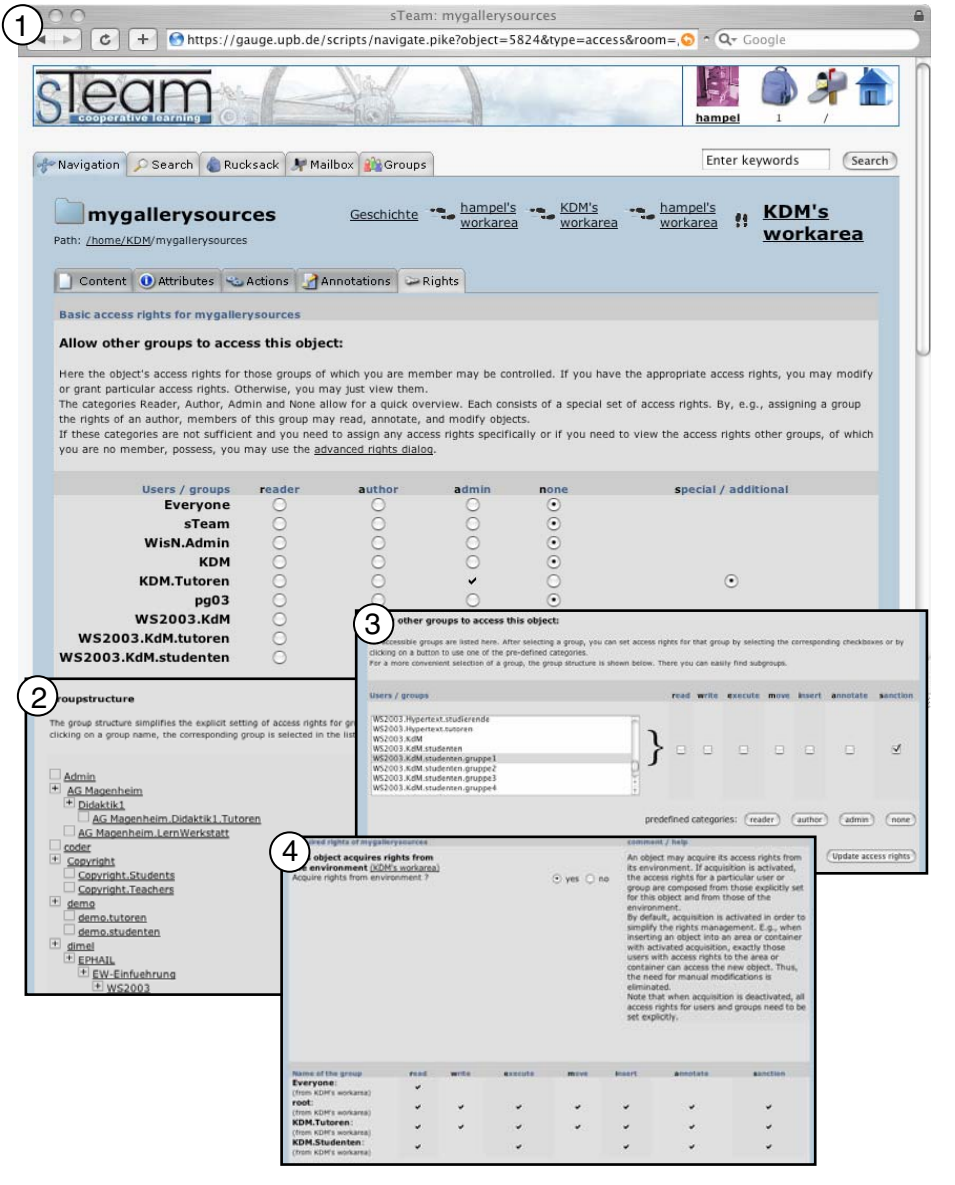



**Fig. 5.** Access rights as implemented in the sTeam-system:
(1) assigning roles to groups and users
(2) delegation of access rights from the environment of an object
(3) using the sanction access right to give the right to administrate access rights
(4) explore the group structure to see inherited access rights form the group structure

Figure 5 shows the user interfaces of the currently implemented access rights model of the sTeam system. As mentioned before the interface implemented here

hides much of the complexity of its underlying access rights model from the user. The upper left screenshot shows the assignment of roles to users. On the left the delegation of access rights from the environment of an object is shown. This feature is activated by default. If a more detailed specification of access rights is needed the upper right dialogue can be used. Here various elementary access rights are presented where also the sanction access rights can be seen. The dialogue on the right allows to browse the hierarchical group structure to interactively explore which access rights are inherited from the group structure.

The following paragraphs document some of the most important findings on the use of our access rights model in cooperative knowledge construction and Computer-Supported Cooperative Learning systems, based on our experience with the $^{open}$sTeam environment: We restricted our self's here especially to findings regarding access rights. More evaluation results are documented in [17].

*Negative access rights are difficult to understand for most users.*

One of the lessons we learned is that negative access rights are difficult to understand for most users. Being able to specify negative access rights also to groups makes it difficult to understand why a granted right may be gone by inheriting a negative access right from another group. It seems to be a good idea to restrict negative rights to the leaves of the inheritance tree and therefore to single users. As a result the actual sTeam platform supports negative rights, but they are not accessible trough the user interface at the moment. Our experiences show that this strategy works quite well.

*Access rights matching the group structure and context are prerequisites for cooperative actions. Documents are only moved or re-arranged if this is possible without additional effort for defining access rights, etc. This means that user rights and authorizations must be suitably preset.*

The second may be most important lesson we learned is that there is somehow a need to reduce motoric effort when supporting collaborative tasks in a CSCW environment. This motoric effort leads directly to the criterion regarding suitable automated definition and presetting of user rights, i.e. *authorizing* as a part of cooperative media functions. Cooperation processes, such as the exchange of documents between learners and teachers (submission of completed assignments) or among different groups of learners, can only be successful if they do not require additional adjustment effort or special precautions with respect to access rights. The approach specifically chosen for the course *Praxis der Systemgestaltung* [15] was to preset all rights to match the group structure with its various component tutorial groups. Documents are given access rights to match the context in which they are created or deposited. If possible, teachers and learners should not – or only under special circumstances – need to modify these rights. All access rights are delegated from the environment of the respective learning environment. – Therefore our goal is to hide access rights form the usrs/learners but on the same time having a powerful access rights model in the background.

As part of the learning process, students should be able to create, arrange and link material, as well as being able to exchange such material and generate shared views on it. To be more specific: this means that material is made available as part of a

course; what students must learn is to relate and reference this material to other sources, and to annotate and structure it in a group context. The core functions of cooperative media can thus be reduced to a series of activities that should be available to all learners, based on the principle self-organization:

These include:

- *creating*, generating, visualizing, symbolizing or modelling arbitrary material in areas as well as generating areas and adding new material
- *deleting*, removing material, annotations and gates as well as areas
- *arranging* and grouping material into units, generating containers and substructures
- *marking/labelling*, marking, highlighting specific elements of material, annotating material
- *linking*, physically connecting area by means of gates, generating reference objects to external material

Only if it is possible to apply such primary media functions and combine them without any additional efforts (e.g. the need to specify access rights) users are willing to cooperatively structure their knowledge spaces. Therefore the simplicity of the access right model or better its features to adapt access rights to collaborative tasks (cf. Fig. 3: Contextual access rights) is one precondition for self organized collaborative spaces.

*The delegation of access rights – sanction right is the key to distributed administration*

Most of today's collaborative systems are still restricted to centralized administration models where administration lies in the hand of one or a group of authorized users. Also in modern LDAP approaches where the user und group structure is stored in a central LDAP directory and distributed to a cluster of collaborative servers the logical structure is centralized. Self administrated features such as the invitation to groups or the totally decentralized administration of parts of the learning environment is hard to archive in such architecture models.

Our experiences show that for really collaborative scenarios where groups are flexible formed and constantly adapted by the learners themselves the access rights model has to support such flexibility. Our sanction access right (see Fig. 5) proved to be a well working concept to archive such flexibility of a decentralized administration.

Therefore the key factor determining the acceptance of a virtual environment for cooperative knowledge organization is the degree to which it proves possible to maintain consistency between real-world groups of learners and their virtual counterparts. Learners must be able to map real-world group structures, such as their tutorial group and their learning groups, to the virtual learning and work environment easily and without extra effort. Experience with the course *Praxis der Systemgestaltung* has shown that learners constantly wish to establish relations between the real-world working group and the virtual group structures. This is evident from typical questions like "Where have you deposited the slide for this tutorial?" or "Can we deposit our notes on this tutorial in a shared area?" In both cases, the knowledge area assigned to

each real tutorial group via its virtual counterpart is the appropriate place for depositing and working on the material forming part of a learning process.

*Access rights to flexible adapt knowledge spaces*

Closely connected with this blending of real and virtual group structures is the adaptability of the environment for virtual knowledge construction to widely differing use scenarios. Experience with different practical implementations of the system has shown that the teachers, too, favour highly diverse forms of information flow and document exchange. In the course *Grundlagen der Informatik für Lehrämter* (basic computer science for teacher education), for instance, very different ways of submitting completed assignments were implemented than in, say, the following *Einführung der Informatik für Medienwissenschaftler* (computer science for media scientists) course. Thus, the key factor determining acceptance of an environment for virtual knowledge organization is its ability to implement and map different forms of cooperation. Accordingly, the sTeam system mainly provides generic functions and procedures for the cooperative handling of material and constitutes as open as possible a platform that can be fleshed out according to the users' needs and preferences. The access rights model which its contextual access rights and the sanction right are important preconditions to archive such a flexibility [21]. The model of being able to handle over the right to grant new access rights allows a completely de-centralized administration of knowledge areas. As the implemented access rights model also controls the ability to include new tools, objects or even new code into a knowledge room the sanction rights is also the key concept to equip rooms with totally new functionality.

## 5 Conclusions

The presented access rights model was designed to be as open and flexible as possible for use in a system of cooperative knowledge spaces. No consideration was given to explicitly negative rights. These can be useful for selectively revoking a user's authorization, but they complicate the model considerably, especially in connection with the inheritance of rights.

The existing kernel architecture in the form of the sTeam system provides a basis for future testing of various models designed for use in virtual knowledge spaces. For instance, based on the concept of the contextual inheritance of rights, rights can be tied to areas only, thus implementing a simple geographical/spatial model of rights setting [3].

Future research will be concerned with selecting suitable inheritance structures for rights and determining the number of required individual rights, but its main focus will be on the new mobility requirements of cooperative knowledge spaces. On this point in particular, future work will need to study and test new demands being made on the access rights model's capabilities (in terms of its clarity and simplicity) for spontaneously emerging knowledge areas (i.e. knowledge areas forming ad hoc net-

works) or for temporary knowledge areas. Initial tests have shown that the chosen model offers the necessary scope here.

## Acknowledgments

Our thanks go to all sTeam system developers, especially Thomas Bopp, Daniel Büse and Ludger Merkens.

## References

1. Bertino, E., Samarati, P., Jajodia, S.: An Extended Authorization Model for Relational Databases. IEEE Transactions on Knowledge and Data Engineering 9(1) 1997, 85–101.
2. Bentley, R., Appelt, W.: Designing a System for Co-operative Work on the World-Wide Web: Experiences with the BSCW System. In: Nunamaker, J.F., Sprague, R. H. (Hrsg.): Proceedings of the HICCS '97, 30th Annual Hawaii International Conference on System Sciences, Vol. IV, January 7-10, Wailea, Hawaii. Washington: IEEE Computer Society Press 1997, 297–306
3. Bullock, A., Benford, S.: An access control framework for multi-user collaborative environments. In: Hayne, S.C. (Hrsg.): Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work (Group99), November 14-17, Phoenix, USA. New York: ACM Press 1999, 140–149.
4. Dourish, P., Edwards, K., LaMarca, A., Salisbury, M.: Presto: An Experimental Architecture for Fluid Interactive Document Spaces. ACM Transactions on Computer-Human Interaction 6(2) 1999, 133–166.
5. Ellis, C.A., Gibbs, S.J., Rein, G.: Groupware: some issues and experiences. Communications of the ACM 34(1) 1991, 39–58.
6. Gavrila, S.I., Barkley, J.F.: Formal Specification for Role Based Access Control User/ Role and Role/ Role Relationship Management. In: Proceedings of the third ACM workshop on Role-based access control, October 22 - 23, 1998, Fairfax, USA, 81–90.
7. Greif, I., Sarin, S.: Data Sharing in Group Work. ACM Transactions on Office Information Systems 5(2) 1987, 187–211.
8. Griffiths, P.P., Wade, B.W.: An Authorization Mechanism for a Relational Database System. ACM TODS 1(3) 1976, 242–255.
9. Haake, J.M., Schümmer, T., Haake, A., Bourimi, M., Landgraf, B.: Supporting Flexible Collaborative Distance Learning in the CURE Platform, Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 1, January 05 - 08, 2004, Big Island, Hawaii.
10. Hampel, T., Keil-Slawik, R.: sTeam – Designing an Integrative Infrastructure for Web-Based Computer Supported Cooperative Learning. In: Proceedings of the 10th International World Wide Web Conference, May 1-5, 2001, Hong Kong, 76–85.
11. Hampel, T. Virtuelle Wissensräume. – Ein Ansatz für die kooperative Wissensorganisation, Universität Paderborn, Fachbereich 17 – Informatik, PHD-thesis, März 2002, in German.
12. Hampel, T., Keil-Slawik, R.: sTeam: Structuring Information in a Team – Distributed Knowledge Management in Cooperative Learning Environments. In: ACM Journal of Educational Resources in Computing 1(2) 2002, 1–27.

13. Hampel, T.: Our Experience With Web-Based Computer-Supported Cooperative Learning – Self-Administered Virtual Knowledge Spaces in Higher Education. In: Proc. of Site 2003 – Society for Information Technology and Teacher Education - International Conference. Charlottesville (Va.), USA: Association for the Advancement of Computing in Education 2003, 1443-1450.
14. Hampel, T., Eßmann, B.: Self-Administered Cooperative Knowledge Areas – Evaluation of the WWW Interface in Terms of Software Ergonomics, In: Harris, D., Duffy, V., Smith, M., Stephanidis, C., Human - Centred Computing – Cognitive, Social and Ergonomic Aspects, Volume 3, Lawrence Erlbaum Associates, Publishers, Mahwah, New Jersey, London, 2003, 729-733.
15. Hampel, T., Keil-Slawik, R.: Experience With Teaching and Learning in Cooperative Knowledge Areas., Proceedings of The Twelfth International World Wide Web Conference, 20-24 May 2003, Budapest, Ungarn, published on CD-ROM, 1-8.
16. Hampel, T., Keil-Slawik, R., Eßmann. B.: Jour Fixe - We Are Structuring Knowledge Collaborative - Structuring of Semantic Spaces as a Didactic Concept and New Form of Cooperative Knowledge Organization. In: Rossett, A. (Ed.), Proceedings of E-Learn 2003, AACE Press, 2003, 225-232.
17. Hampel, T.: Computer Supported Cooperative Learning - a Set of Theses, Society for Information Technology and Teacher Education International Conference, Vol. 2004, Issue. 1, 2004, pp. 937-944.
18. Kawell, L., Beckhardt, S., Halvorsen, T., Ozzie, R., Greif, I.: Replicated Document Management in a Group Communication System. Proceedings of the 2nd International Conference on Computer-Supported Cooperative Work (CSCW'88), September 26-28, 1988, Portland (Or.), USA. SIGCHI/SIGOIS ACM 1988, reprinted in: Marca, D., Bock. G. (Hrsg): Groupware: Software for Computer-Supported Cooperative Work. IEEE Computer Society Press 1992, 226–235.
19. Lampson, B.: Protection. In: Proceedings of the 5th Princeton Conference on Information Sciences and Systems, Princeton, 1971. Reprinted in ACM Operating Systems 8(1) 1974, 18–24.
20. Lampson, B., Abadi, M., Burrows, M., Wobber, E.: Authentication in Distributed Systems: Theory and Practice. ACM Transactions in Computer Systems 10(4) 1992, 265–310.
21. Rubart, J., Hampel, T.: Structuring Cooperative Spaces: From Static Templates to Self-Organization. In: Hicks, D. L. (Ed.): Metainformatics. International Symposium, MIS 2003, Springer-Verlag, 2004, 119–125.
22. Satyanarayanan, M.: Integrating Security in a Large Distributed System. ACM Transactions on Computer Systems 7(3) 1989, 247–280.
23. Sikkel, K.: A Group-based Authorization Model for Computer-Supported Cooperative Work. GMD – Forschungszentrum, working draft, GMD 1055, März 1997.
24. Shen, H., Dewan, P.: Access Control for Collaborative Environments. In: Turner, J., Kraut, R. (Hrsg.): Proceedings of the Conference on Computer-Supported Cooperative Work – CSCW92, October 31 - November 4, Toronto, Canada. New York: ACM Press 1992, 51–58.
25. Stiemerling, O., Wulf, V.: Beyond "Yes and No"- Extending Access Control in Groupware with Awareness and Negotiation. In: Darses, F., Zaraté, P.P. (Hrsg.): Proceedings of the Third International Conference on the Design of Cooperative Systems (COOP´98), Vol. I, 26.-29.5.1998, Cannes, France, INRIA, 1998, 111–120.

26. Trevor, J., Rodden, T., Mariani, J.: The Use of Adapters to Support Cooperative Sharing. In: Furuta, R., Neuwirth, Ch. (Hrsg.): Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'94), October 22 – 26, Chapel Hill, USA. New York: ACM Press 1994, 219–230.
27. Zhang, Z., Haffner, E., Heuer, A., Engel, T., Meinel, Ch.: Role-based Access Control in Online Authoring and Publishing Systems vs. Document Hierarchy. In: Proceedings on the Seventeenth ACM Annual International Conference on Computer Documentation, September 12-14, 1999, New Orleans, USA, 193–198.

# Flexible Notifications and Task Models for Cooperative Work Management

Jessica Rubart[1] and Helge Richter[2]

Fraunhofer IPSI, Dolivostrasse 15, 64293 Darmstadt, Germany
[1] rubart@ipsi.fraunhofer.de, info@jessicarubart.de
[2] mail@helge-richter.de

**Abstract.** Knowledge intensive cooperative work requires emergent workflow management. Participants interact with the workflow engine and jointly redefine and activate workflow structure. To improve the usability of such systems we present reconfigurable notification mechanisms as well as shared task models that can be used from diverse clients at the same time focusing on different kinds of visualization and navigation.

## 1 Introduction

Conventional workflow management systems (WfMSs) and architectures [11] focus on support for automatic enactment of asynchronous cooperative work, where the flow of work is usually predefined. *Knowledge intensive cooperative work*, however, is neither recurring nor prescribed in detail. Rather, work plans evolve as the work progresses. *Interactive process models* [4, 5] address knowledge intensive work by involving users in contextual model interpretation. Interactive process models are created and updated by the project participants to reflect evolving plans.

In [8] we argue that interactive model activation does not only include the interaction between the participants and the workflow engine, but also the interaction among the participants themselves. Cooperation-aware synchronous groupware systems supporting informal and creative interactions between participants need to be integrated with flexible work management. This enables joint (re)definition of emerging activity breakdown structures and their flexible interactive activation.

During the evaluation of this approach in the EXTERNAL project [2] it was suggested to open the integrated environment to standard communication means, e.g. to add more e-mail notifications when changes happen. In [7] a survey on workflow management systems identified the need to support reconfigurable notification mechanisms with respect to occurring changes. Additionally, contextual navigation possibilities through visual workflow models in an easy to access web-based environment were required.

In this paper, we present reconfigurable notification mechanisms as well as shared task models that can be used from diverse clients at the same time focusing on different kinds of visualization and navigation. Our approach builds on shared hypermedia workspaces that are configured to model workflow structure [8, 9] and addresses the

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 32 – 41, 2004.

above mentioned requirements. It focuses on communication-oriented work management as shared visual models that evolve over time as well as explicit notifications support the communication among the participants.

From the *metainformatical* view [6], taken in this community, work management systems support an organization's primary task and, thus, belong to the B level of Engelbart's ABC model of organizational improvement. The usability of these systems can be improved by configurability and easy to access contextual visualization and navigation means. Therefore, improving the usability of work management systems can be considered C level work.

## 2 Knowledge Intensive Cooperative Work

Within knowledge intensive cooperative work problems are incompletely understood. While solving the problems knowledge workers develop a common understanding about them. Success depends on joint problem solving capabilities and creativity of the knowledge workers [5]. Work management support needs to be very flexible as work plans evolve.

Table 1 shows two extremes of workflow management systems. Today's WfMSs usually focus on labor or capital intensive work. They provide support for automatic enactment of processes, which typically are repetitive and modeled by experts. In addition, workflow definition and enactment are clearly separated phases within the workflow management lifecycle. Emergent WfMSs, in contrast, focus on knowledge intensive work. Participants are involved in interpreting the workflow and can jointly redefine, negotiate, and perform cooperative activities [5, 8]. Those processes are usually unique. Participants can adapt them during activation and thus articulate their work at any time.

**Table 1.** Two Extremes of WfMSs

| *Today's WfMSs* | *Emergent WfMSs* |
|---|---|
| Labor/capital intensive work | Knowledge intensive work |
| Automatic enactment | Collaborative interactive activation |
| Repetitive processes | Unique processes |
| Modeled by experts | Modeled by participants |
| Separated definition and enactment | Integrated articulation and activation |

Most work integrates knowledge intensive parts. We applied collaborative interactive activation of emergent workflows in the EXTERNAL project and had positive results [2, 5]. Suggestions for improvement, in particular with respect to knowledge intensive work, where change is more the norm rather than the exception, include [2, 7]:

- Reconfigurable notification means to improve the awareness of cooperative work management
- Contextual navigation possibilities through visual workflow models in an easy to access web-based environment

## 3 Flexible Notifications and Task Models

Our approach builds on shared hypermedia workspaces that are configured to model workflow structure [8, 9]. These workspaces support cooperative modeling of emerging workflows and their flexible collaborative interactive activation.

In the following we describe how shared hypermedia-based task models can be used from diverse clients at the same time focusing on different kinds of visualization and navigation. Afterwards we present integrated notification mechanisms that open the environment to standard communication means. Finally, we describe a way to provide contextual navigation possibilities through visual workflow models on the web.

### 3.1 Shared Hypermedia-Based Task Models

Tasks are modeled as shared hypermedia composites [8] to enable synchronous cooperation and complex task structures. The sharing is a built-in characteristic of the hypermedia objects. Every hypermedia object has a unique identifier and is potentially accessible from all participants. Concurrency control mechanisms take care of consistency. Notification mechanisms can be seen on different levels. On the low level of object sharing notification mechanisms support the consistency of an object and its different representations in a distributed system. For synchronous cooperation, in order to provide immediate feedback about the interactions of other users, notifications about changes on shared objects are supported. In addition, persistency is provided for shared objects to support asynchronous cooperation, continue synchronous work later, and support transitions between asynchronous and synchronous cooperation.

Our data model uses special node and link types for representing workflow structures explicitly. Task objects are composite objects and can thus contain other nodes and links in order to model nested structures. A system implementing shared hypermedia composites already knows about changes on task structures and attributes. Thus, changes on tasks related to end user notifications can easily be identified.

Fig. 1 shows our architecture supporting different kinds of visualization and navigation with respect to the same shared hypermedia-based task models. The greyly marked *CoWom* (*communication-oriented workflow management*) components contain a specific shared hypermedia-based task model that is based on top of the generic hypermedia model of XCHIPS4KM[1] [9] and can thus be used in associated shared hypermedia workspaces. Instances of such task models are replicated between a server and connected XCHIPS4KM clients accessing them. At the same time specific task servlets reside on the server side and can access and manipulate those models. Thus, a standard web client can work on the same task models by interacting with those servlets.

[1] XCHIPS4KM adds a configuration environment to XCHIPS [8] and is implemented on top of the DyCE groupware framework [10].

In addition, the server side contains a notification service focusing on end user notifications. This services observes specific task models in order to send notifications based on configured changes, such as a reached deadline (cp. next section). It can also be used from *CoWom* clients as any other service in the system and from task servlets to configure notifications in the system.

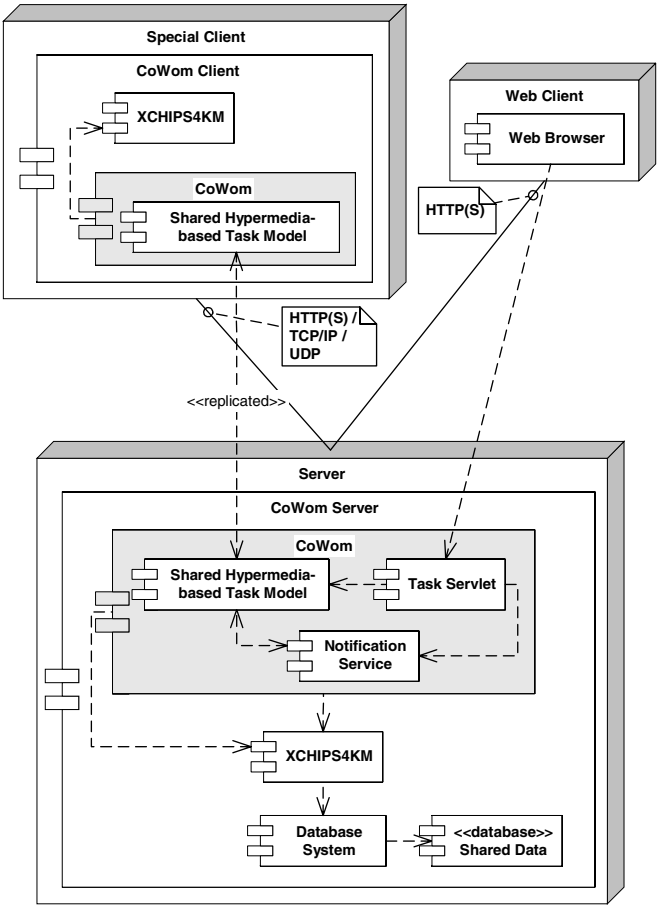



**Fig. 1.** Architecture

### 3.2 Notification Mechanisms

In our approach, end users are able to configure different kinds of notification types based on changes in dynamic workflows, such as status changes, structural changes, or changes related to the notification configuration itself.

Fig. 2 shows example notifications by e-mail (top part), news feed (middle part), and instant messaging (bottom part).

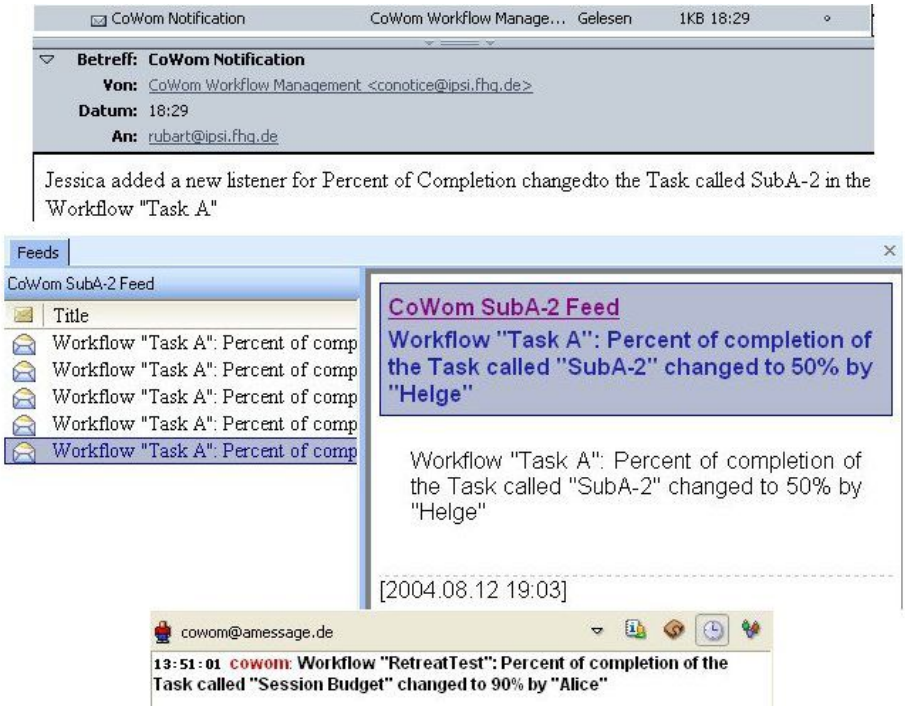



**Fig. 2.** Notifications by E-Mail (shown in Netscape[2]) , News feed (shown in Abilon News Aggregator[3]), and Instant Messaging (shown in Miranda IM[4])

E-mail is a traditional well-known asynchronous communication mechanism that is widely used.

RSS news feeds are increasingly used on websites in order to provide news and events. RSS[5] (*RDF Site Summary*) is an RDF[6] (*Resource Description Framework*) vocabulary for describing news and events. With respect to workflows and tasks news feeds can provide histories of configured events for users and groups of users. Without having to view the different workflows or tasks, news feeds provide the most interesting changes at a glance.

Instant messaging focuses on synchronous communication and is starting to get used seriously in business life. Messages are transferred immediately between participants. Some people criticize the *instant* nature of this medium as being *annoying*, but usually this is a matter of configuration. For instance, people can indicate whether they are available or occupied, whether any, just urgent, or even no messages should pop up. Usually, messages can be read later when not turned up instantly.

[2] http://www.netscape.com/
[3] http://www.abilon.org/
[4] http://www.miranda-im.org/
[5] http://web.resource.org/rss/1.0/
[6] http://www.w3.org/RDF/

These examples of notification types can, of course, be extended, for instance by messages to mobile phones based on SMS (*Short Message Service*) or MMS (*Multimedia Message Service*).

Fig. 3 shows fundamental differences of some notification types. While e-mail has its place as the average of response time and degree of disturbance, other notification types do have their own areas of application. News feed, for example, has a similar response time as snail mail, but a lower degree of disturbance. Of course, in order to make available those notification types to the notification service corresponding APIs need to be available and integrated.

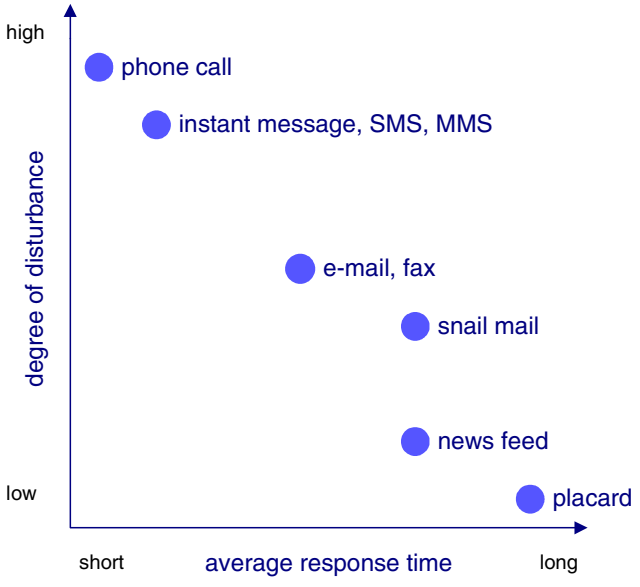



**Fig. 3.** Differences of some Notification Types

Changes on tasks that are *big* enough to be a potential basis for end user notifications should appear in configuration dialogs. Fig. 4 shows an example configuration dialog taken from a *CoWom* web page, which presents a task description and supports its manipulation.

The upper part shows configuration options for the currently logged in user. A *listener type* represents an important change on the associated task, such as *status changed*. Next to *notify via* the notification type can be selected, such as instant messaging via *Jabber*[7]. Jabber is an XML-based messaging protocol that is supported by several instant messengers. The lower part of Fig. 4 shows actors assigned to the current task and their configured notifications. This is one way of how to represent such configurations in a user interface. In addition, not only web-based clients, but also specific *CoWom* clients provide such functionality.

[7] http://www.jabber.org/


**Fig. 4.** Example of Configured Notifications

### 3.3 Web-Based Visual Task Models

To provide contextual visual workflow models in an easy to access web-based environment SVG[8] (*Scalable Vector Graphics*) seems a good choice because it can be generated for every task and visualized in browsers. Fig. 5 shows an example web-based visual task model using SVG. It has three parts: The current task is shown in the middle; its predecessors are visible on the left side and its successors are presented on the right side. The icons provide awareness about task related information, such as its status or whether it's assigned to the current user.

Navigation is also possible through this visualization. Clicking on such a task visualization of a predecessor or successor task generates a *CoWom* web page, which presents the details of the selected task and supports its manipulation. In addition, the SVG-based contextual view on the selected task is presented, i.e. the selected task is shown in the middle. Clicking on the middle task will show the contents of that task, i.e. its subtasks. Navigating to one of those subtasks provides its details and the SVG-based contextual view.

Such easy to access web-based visual task models complement special *CoWom* clients that are more complex and also support synchronous cooperation. Special *CoWom* clients can be started via the web by using technologies, such as Java Web Start[9], but are not based on a markup language. They can provide more sophisticated

[8] http://www.w3.org/TR/SVG/
[9] http://java.sun.com/products/javawebstart/

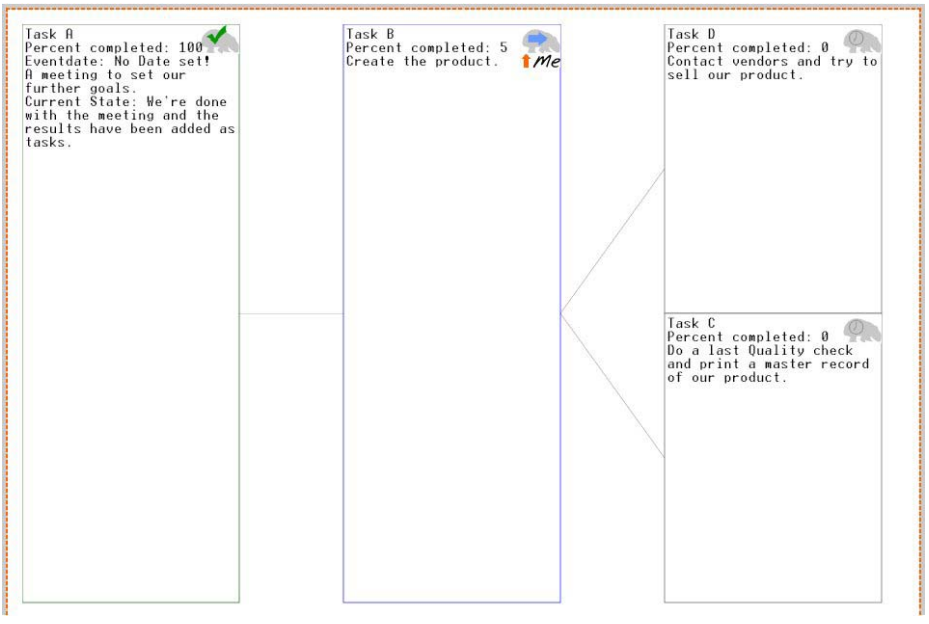



**Fig. 5.** A Visual Task Model with SVG

cooperative user interfaces, such as the workflow modeling and activation user interface of XCHIPS [8]. With this approach both kinds of clients and user interfaces can be provided and combined.

## 4 Related Work

Traditional WfMSs, such as WWWorkflow [1], focus on support for automatic enactment of asynchronous cooperative work. They are mainly used to automate routine procedures, which can be completely prescribed.

Ad hoc WfMSs, such as Freeflow [3] and WORKWARE [4, 5], address flexible workflow management by allowing users to participate in interpreting the workflow. However, they lack support for synchronous cooperation to enable joint redefinition of emerging workflow structure that includes the negotiation of joint work processes and the facilitation of synchronous access to shared information resources providing immediate interactive feedback and awareness.

Both kinds of WfMSs usually support, if any, just e-mail notifications. To our knowledge there is no workflow management approach supporting reconfigurable notifications, which are particularly important with respect to emergent workflows.

Concerning architectures for structural computing environments [12, 13] a foundation services layer has been proposed that provides basic infrastructure services like versioning and notification control. Such notification control focuses on event notifications in the system rather than on notifications for end users (cp. section 3.1).

## 5 Conclusions

We have presented an approach supporting communication-oriented cooperative work management. It improves particularly knowledge intensive cooperative work management (but also traditional) by providing reconfigurable notification means and flexible visual task models that can be used from diverse clients at the same time focusing on different kinds of visualization and navigation. It builds on the approach presented in [8] that integrates flexible workflow management with cooperation-aware synchronous groupware. The same shared hypermedia objects, such as tasks and actors, are manipulated from the different clients. Integrated notification mechanisms open the environment to standard communication means and thus improve the acceptance of such dynamic cooperative workflow management solutions.

In our future work we want to generalize the notification mechanisms and provide a nonspecific notification framework. This can be applied to very different application domains, such as to meta-modeling activities presented in [9], which support the definition and evolution of shared modeling languages, or to structural computing environments [12, 13] to support end user notifications based on structural changes managed by different structure services.

## Acknowledgments

We would like to thank the attendees of the Metainformatics Symposium 2004 in Salzburg as well as the anonymous reviewers for their valuable feedback.

## References

1. Ames, C.K., Burleigh, S.C., and Mitchell, S.J.: *A Web Based Enterprise Workflow System*, In: Proceedings of GROUP'97, ACM Press (1997)
2. Chrysostalis, M., Hildrum J., Krogstie, J., Scagno, G., and Stromseng, K.: EXTERNAL WP9 – Deliverable D5: Use Case Evaluation Report, available at http://www.external-ist.org/ (2003)
3. Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P., and Zbyslaw, A.: Freeflow: Mediating Between Representation and Action in Workflow Systems. In: Proceedings of CSCW '96, ACM Press, 1996.
4. Jørgensen, H.D. Interaction as a Framework for Flexible Workflow Modelling. In: Proceedings of Group 2001. Boulder, USA: ACM Press (2001)
5. Jørgensen, H.D. Interactive Process Models. PhD-thesis, Norwegian University of Science and Technology, Trondheim, Norway (2004)
6. Nürnberg, P.J.: Building Metainformatical Bridges. In: Metainformatics. International Symposium, MIS 2002, Springer-Verlag, (2003) 6-8
7. Richter, H.: CoWom – Ein kommunikationsorientiertes Workflow-Management-System. Diploma thesis at Darmstadt University of Technology, (2004, in German)
8. Rubart, J., Wang, W., Haake, J. M. Supporting Cooperative Activities with Shared Hypermedia Workspaces on the WWW. In: Alternate Track Proceedings of WWW 2003, MTA SZTAKI (2003)

9. Rubart, J., Wang, W., and Haake, J. M. A Meta-Modeling Environment for Cooperative Knowledge Management. In: Metainformatics. International Symposium, MIS 2002, Springer-Verlag, (2003) 18-28
10. Tietze, D.A.: A Framework for Developing Component-based Co-operative Applications. In: GMD Research Series No. 7/2001, ISBN: 3-88457-390-X (2001)
11. WfMC, Workflow Handbook 2001, ed. L. Fischer. 2000, Lighthouse Point, Florida, USA: Workflow Management Coalition, Future Strategies Inc. (2000)
12. 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
13. Wiil, U.K., Hicks, D.L., and Nürnberg, P.J.: An Agenda for Structural Computing Research. In: Metainformatics. International Symposium, MIS 2004, Springer-Verlag, (2004, this book)

# Managing Ontological Complexity: A Case Study

Peter J. Nürnberg and Svetlana Krestova

Department of Software and Media Technology,
Aalborg University Esbjerg,
Niels Bohrs Vej 8
DK-6700 Esbjerg, Denmark
{pnuern, sak}@cs.aue.auc.dk

**Abstract.** Ontologies represent a widely accepted method for modelling structured knowledge spaces. Ontologies are particularly useful in modelling corporate or collective knowledge spaces. As such, they provide a vehicle for codifying the collected experience, best practices or common agreement of communities. Nonetheless, in practice, such knowledge must be tailored by practitioners to meet the challenges at hand. While tools for the construction of ontologies abound, we have found that many users are still reliant on human judgement instead of computational support when ontological knowledge must be tailored, personalized, customized and/or applied. In this paper, we examine some of the circumstances surrounding this state of affairs, and contemplate possible roles for computational support in these undertakings. We do this through based on our experiences with the Multilingual Dictionary of Lexicographical Terms (MDLT), a prototype of a linguistic database.

## 1 Introduction

Recently, more and more communities have started to implement ontologies in their studies and create their own domain-specific or general purpose ones. Scientists use ontology for describing concepts and relationships that can exist for an agent or community of agents.

Ontology creators use different levels of complexity and address different kinds of problems. Some communities (e.g., computer scientists) generate application-focused databases from large ontologies and study optimization schemes to facilitate high quality sub-ontology extraction or the development of a translation approach to portability preserving the computational efficiency of implemented systems. Other groups of scientists find the problem of ontology evolution fundamental. They investigate questions of how to manage multiple versions of an ontology and how to represent differences and transformations between ontologies. Even when an entire community is studying one problem, there still can be different opinions about the same question [1, 2].

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 42–50, 2004.

## 2 Lexicography

The common problem in ontology building for every community is that there are few hard and fast rules for creating an ontology – every specialist must follow personal experience during the work. There is a possibility that members of the same community belonging to different languge cultures have different ways of concept structuring in the same ontologies; misunderstandings because of the translation problems; or, non-existing language concepts. All of these problems were discoved earlier by terminologists and lexicographers. It is easy to illustrate them from already existing experiences.

For many years lexicographers were creating dictionaries of different types: monolingual; multilingual; encyclopedias; general purpose dictionaries; and, terminological dictionaries. The ones we address here about are specialized or terminological dictionaries. This type of dictionary is the most interesting for us because the process of making such dictionaries includes ontology building and concept relationships analysis. To illustrate aforementioned problems, we use the Multilingual Dictionary of Lexicographical Terms (MDLT), an online multilingual, terminological dictionary currently under development.

Lexicography, as any specialized branch of science has its own history, theory, and practice. For a long time lexicography was understood in a narrow sense – as an applied part of linguistics. But nowadays, with the very rapid advances in technologies and formats of reference materials, partially after the development of electronic media and global computerization, lexicography was recognised as a global academic domain of reference science. This has reinforced and accelerated the growth of lexicography as a separate branch of science. We feel that lexicography now merits its own dictionary reflecting its newly acquired first-class status.

In the twentieth century, printed versions of dictionaries were reedited and even transformed into electronic form and placed on the Web. These were the first steps towards corpus and cyber lexicography. Many well-known publishing houses have started to produce dictionaries on CD and place information on the Internet. This has changed the structure and composition of the material within the dictionary and search strategies. It made the access to the information easier and the speed of the knowledge operation faster.

## 3 Multilingual Dictionary of Lexicographical Terms

One example of a new tendency in dictionary making is the Multilingual Dictionary of Lexicographical Terms (MDLT). The MDLT is a prototype of a linguistic database avaliable on the Internet. The main goal of this project is to provide all users, especially student-linguists, with a tool for increasing lexicographical competence. We based our project on an investigation of users needs and demands. Through our rich entry structure (including translation equivalents, set phrases, synonyms, antonyms, etc.) the system combines large volume with quality of information and convenient query. Our initial motivation was a lack of dictio-

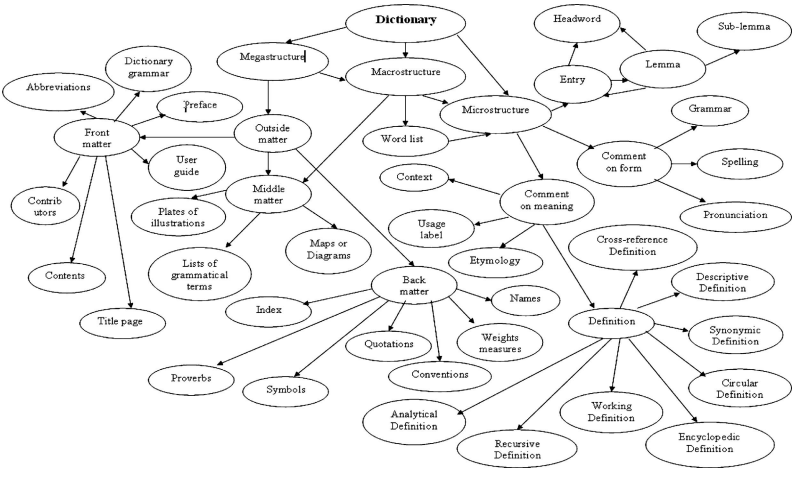



**Fig. 1.** Part of the conceptual system for the subject field of lexicography

naries that serve to underline differences between Russian and English terms in the subject field of lexicography. Therefore, our dictionary contains entries in two languages – Russian and English. However, our system can also support other European languages such as German, French, Italian, Danish, etc.

The first and most important step toward creating an LSP dictionary (what was proved by our work) is building up a conceptual system of the subject field under study. As an example we will take a piece of our conceptual system lexicography illustrating the dictionary structure (see figures below). It is important to state the part-whole relationships between the concepts in a given terminological structure before describing them in the dictionary.

This conceptual system served as a systematic listing of the specialized terminology that helped ensure that no terms and/or categories were missing from our dictionary. The system also helped in searching and defining the hierarchy of definite conceptual structure, common to all texts in a given subject field.

For building up a conceptual system, it is necessary to have lists of concepts chosen from different sources. To make the dictionary reliable, the sources must be selected carefully. In our case, we took all well-known existing dictionaries of lexicographical terms (including [3, 4, 5, 6, 7]) and extracted all items out of them. This resulted in a list of 2697 terms. The next step was to look through the recently published texts (i.e., published within the last five years) on similar subjects to identify additions to lexicographic terminology. This resulted in an expansion of our list to 2828 terms (i.e., an additional 131 items were added). This shows that lexicography is continually developing, underlining the need to create a dictionary that will be able to be updated and enlarged – an electronic dictionary.

Because the dictionary is still under construction, the current version contains 23 totally developed entries in English (lexicography, dictionary, dictionary

structure, megastructure, front matter, outside matter, back matter, macrostructure, wordlist, microstructure, headword, entry, lemma, sub-lemma, comment on form, comment on meaning, labels, spelling, pronunciation, grammar, definition, etymology, usage label) and 22 in Russian (лексикография, словарь, структура словаря, мегаструктура, введение, приложение, макроструктура, словник, микроструктура, заглавное слово, входная единица, лексема, добавочная семантико-функциональная характеристика, информационная категория, орфографическая помета, орфоэпическая помета, грамматическая помета, этимологическая помета, стилистическая помета, частотная помета, дефиниция, контекст) describing the dictionary structure.

The metalanguge was borrowed from the aforementioned sources (lexicographical dictionaries and text).

Lexicographers need to keep in mind the structure of the dictionary they are constructing. This means paying careful attention to the dictionarys megastructure, macrostructure and microstructure. Megastructure always includes outside matter and a macrostructure (i.e., a wordlist organized either in the alphabetic or systematic order). Microstructure varies according to purpose. We chose a users perspective to set up the content of our article structure.

We determined that user-lexicographers in general would like to see articles with the following components: definition (preferrably a descriptive or synonymic definition); context; labels; and, synonyms. Therefore, for each entry, our dictionary provides: a headword; definition; grammar and phonetic labels; context; and, related terms. Terms also have associated sound files with voiving by native speakers. Additionally, elements of one entry can be hyperlinked with other entries. For example, the article dictionary has hyperlinks to the articles: reference work; words; reference book; and, wordbook. Hyperlinks provide users with quick access to the data stored in the dictionary.

## 4 Problems of Interpretation

It is very important to concentrate on the user perspective while creating a dictionary. Every dictionary has to respond to users needs and demands. A lexicographer should realize that different users have different reasons for using a dictionary. That is why the content of MLDT and the detailed description of the entries serve many purposes. For translators, the system has term equivalents in different languages and related terms, which may help them make adequate translations from one language into other(s). Beginners can find many interesting facts in the introductory part, which is available in both English and Russian. Transcriptions will help users to pronounce terms correctly. Contexts show how to use entries in better ways. We provide scientists with detailed concept descriptions or definitions with references to sources used. Related terms can show the placemant of definite concepts in the lexicographical conceptual system and state the relationship with the other concepts in the system. All the aforementioned features make this dictionary reliable and user friendly.

Another problem, also common in dictionary making, is different ways of concept structuring in different languages. In every country, the scientific communities have their own way of looking at the same things and because of that they have slightly different terminology as well.

The structure of the ontologies they have built differes much especially in relationship between the concepts described within it. A good example of structural differencies could be seen while comparing two sub-ontologies of a dictinary microstructure in English and Russian languages (see Fig. 2).


**Fig. 2.** English and Russian conceptual structures for the term *microstructure*

Upon inspection, it is clear that their structures are different. This means that a community of lexicographers look at the dictionary structure problem from different perspectives. For example, microstructure according to English lexicographers includes three main elements: headword (or lemma); comment on form (i.e., grammar, spelling and pronunciation labels); and, comment on meaning (i.e., definition, context, etymology and usage label). According to the Russian school of lexicography, dictionary microstructure has four main elements: headword; definition; context (or verbal illustration); and, labels (i.e., comment on form and meaning excluding definition and context).

The next important problem in dictionary making is finding translational equivalents in two or more languages. It is possible that the concept in one languge could not be transfrered into another languge, because its translational equivalent does not exist. This is illustrated by Fig. 3.


**Fig. 3.** English and Russian conceptual structures for the term *macrostructure*

In English, the concept megastructure includes two sub-sections: outside matter (i.e., frontmatter, middlematter and backmatter); and, macrostructure. In Russian, the concepts outside matter, front matter and middlematter do not exist. One can talk only about backmatter, because the additional information in Russian dictionaries is normally kept at the back; the parts such introduction, user's guide, wordlist, guide to pronunciation and abbreviations form their own parts in the dictionary structure. This is a good example how people in the same community have different ways of differentiating concepts and ontologies in general.

Sometimes even if the translational equivalent to a certain amount of concepts exist in another language, they still could have different meanings. For example, when translating the term middlematter from English into Russian, some individuals in the Russian school of lexicpgraphy would claim that middlematter is everything that is in the middle of a dictionary – a wordlist, including the tables and illustrations inside it, if any. If there are no tabels and illustrations in the dictionary, then it is the wordlist itself, which is the macrostructure of

a dictionary. However, if you will look at the ontology of megastructure given above, it is absolutly clear that English lexicographers define middlematter differently. For them, it is separate from a macrostructure field, which includes list of grammatical terms, maps and diagrams put in the middle of a wordlist in a reference book. This example shows that there are still many problems in ontology creation and a dictionary making processes.

The aforementioned problems are true not only when comparing two ontologies on two or more different languages; they can be found even within one languge society. One of the brightest examples is the scientific attitude to slang. If one considers a general purpose dictionary, one will find some well known slang words, such as ass (arse), dick, fuck, shit, bastard, etc. Linguists disagree as to whether slang should be included in general purpose dictionaries – ones that describe the whole language culture, or whether it should be avoided and included only in slang dictionaries, or even if it should be defined at all.

Some linguists think that slang is too abusive and should be avoided in the dictionries that describe national language. However, if we create a dictionary that reflect the way we speak, and we use slang in everyday life, then what should we do? Another group of linguists thinks that slang is a part of our language and our culture and if people use it in their language it should be described in the dictionary together with any other common words existing in the language. This question is still open. Some people think that defining slang is easy, but what do we call slang? Slang includes not only euphamisms (the abusive lexic), but also a language of small communities like students, lawyers, computer scientists and so on. Should we compare this case with general purpose and special purpose dictionaries? It is true that some professional terms from different branches of science, like biology, medicine, chemistry, computer science, linguistics, mathematics, etc., were included into general purpose dictionaries because we use them in our everyday life. If we include terminology in the academic dictionary, why should we not include slang?

## 5 Computer Support

Human judgement, as illustrated above, is still and will always be a fundamental component of ontology construction. With this in mind, we feel that tools that allow researchers to express divergent opinions and emergent understandings are well-suited to supporting ontological work. Many hypermedia and structural computing tools meet the needs of ontology workers by allowing flexible and dynamic knowledge representations. Hypermedia tools complement other technologies that focus on automatic ontology construction (e.g., [8, 9]).

The Construct project [10] has been under development for the past five years. It is comprised of a suite of hypermedia tools that are designed for expressing incomplete, inconsistent, and/or dynamic knowledge. Different tools in the Construct suite focus on different *structural domains*, or types of knowledge structures. There is currently support within Construct for navigational, argumentation, spatial, metadata, and taxonomic structures. We are currently

investigating adding support for knowledge structures used by lexicographers in their work processes. Because of Construct's focus on tools supporting the development of new structural domains, there is relatively low overhead in the implementation of such support. Our work is therefore focused on detailed observation of the lexicographic work process and the resultant design of structure-based hypermedia tools.

Even with the current set of tools, however, we have noticed that hypermedia tools can provide significant advantages to lexicographers. Argumentation structures, for example, allow the construction of spaces capturing rationale behind decision processes. Navigational tools (i.e., tools that provide a "hyperlinking" interface to traversal of associations among knowledge items) can also provide an intuitive way of constructing and communicating complex spaces.

## 6 Conclusions

In this paper, we have considered lexicographic work as a prototypical example of ontological work. We have shown that, even though automatic ontology construction tools may be useful in certain circumstances, they are insufficient alone for supporting knowledge workers enagaged in ontology construction. Instead, we also need tools that allow workers to capture inconsistent and/or incomplete knowledge, and allow multiple versions of concpetual structuring to underlie the same system. Our initial results have shown that hypermedia tools are well-suited to the task of helping knowledge workers represent and manipulate the complex spaces that underpin ontologies.

## References

1. Kang, D., Xu, B., Lu, J., Wang, P., Li, Y.: Extracting sub-ontology from multiple ontologies. In: Proceedings of the OTM Workshops 2004. Volume LNCS 3292., Springer Verlag (2004) 731–740
2. De Leenheer, P.: Revising and managing multiple ontology versions in a possible worlds setting. In: Proceedings of the OTM Workshops 2004. Volume LNCS 3292., Springer Verlag (2004) 798–809
3. Bergenholtz, H., Cantell, I., Fjeld, R., Gundersen, D., Jónsson, S., Mikkelsen, H.K., Svénsen, B.: Nordisk Leksikografisk Ordbok. Universitetsforlaget, Oslo (1993)
4. Burkhanov, I.: Lexicography: A Dictionary of Basic Terminology. Wydawnictwo Wyzszej Szkoly Pedagogicznej, Rzeszów (1998)
5. Hartmann, R.R.K., ed.: Lexicography. Principles and Practice. Academic Press, London (1983)
6. Hartmann, R.R.K., James, G.: Dictionary of Lexicography. Routledge, London (1998)
7. Kipfer, B.A.: Wordbook on Lexicography: A Course for Dictionary Users with a Glossary of English Lexicographical Terms. University of Exeter (1984)
8. Alani, H., Kim, S., Millard, D.E., Weal, M.J., Hall, W., Lewis, P.H., Shadbolt, N.R.: Automatic ontology-based knowledge extraction and tailored biography generation from the web. Technical Report Equator-02-049, University of Southampton, Southampton, UK (2002)

9. Mena, E., Illarramendi, A., Goñi, A.: Automatic ontology construction for a multiagent-based software gathering service. In: CIA '00: Proceedings of the 4th International Workshop on Cooperative Information Agents IV, The Future of Information Agents in Cyberspace, Springer-Verlag (2000) 232–243
10. Wiil, U.K.: Lessons learned with the Construct development environment. In: Proceedings of the First Metainformatics Symposium. Volume LNCS 2641., Springer Verlag (2002) 9–17

# Looking Beyond Computer Applications: Investigating Rich Structures

Claus Atzenbeck and Peter J. Nürnberg

Department of Software and Media Technology,
Aalborg University Esbjerg,
Niels Bohrs Vej 8, 6700 Esbjerg, Denmark
{atzenbeck, pnuern}@cs.aaue.dk

**Abstract.** Spatial structure supporting applications offer an abstract level of what can be found in the real world. However, in many systems, objects are aligned straight, rotation is not possible, they can be resized easily and can hold more text than is visible on the screen. Paper and structures created with paper seem to be more limited: Straight alignment is not possible without spending much time; paper can hardly be resized without damaging it; and piles may fall down if they become too tall. However, a closer look shows that paper structures offer much more attributes and dependencies than any current spatial structure supporting application. In this article, we compare paper structures to a selection of computer applications. We argue that the observed small additions with paper carry information which improves finding and organizing.

## 1 Introduction

An important task in knowledge work is structuring knowledge. This supports the ability to exchange knowledge, easy traversal, and fast retrieval. To extend the capability of the human mind, several tools have been created. A long time ago, people started to write information on paper. They compiled scriptures and cross-referenced them. Later, machines were designed to offer a higher level of structuring.

In the last century, Vannevar Bush described his idea of how a machine can augment the human mind and described *Memex* [4]. The name is an abbreviation for *Memory Extender*. This machine was designed to "build a trail of ... items" which can be followed again at any time later [4]. Decades later, this idea had a very strong influence on a whole field of research: hypertext [27]. To Ted Nelson, who coined the term, hypertext was seen as "a body of written or pictorial material interconnected in such a complex way that it could not conveniently be presented or represented on paper" [22]. The overall goal was to extend the human mind as Douglas Engelbart, another hypertext pioneer, has formulated:

> "By 'augmenting human intellect' we mean increasing the capability of a man to approach a complex problem situation, to gain comprehension to suit his particular needs, and to derive solutions to problems." [10]

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 51–65, 2004.

During the following years, several hypertext systems were developed. All were based on a node–link model. In order to unify the terminology and give a standard model for such systems, the *Dexter Hypertext Reference Model* [13] was developed. This kind of hypertext is currently known as *navigational* or *associative* hypertext.

Later, special structure types were brought to the hypertext community, such as taxonomies [31] or argumentation structures [7, 8]. Influenced by two systems, *gIBIS* (argumentation support) and *NoteCards* (card metaphor with associative links), the first so called spatial hypertext system *Aquanet* [20] was developed. Originally, its developers focused on argumentation support rather than on how people use paper cards in the real world. However with *Aquanet*'s successor *VIKI* [21] and *VKB* [26] the latter mentioned aspect became more important. Differences and similarities between spatial hypertext applications and generic visualization tools have been discussed recently [18].

Beside spatial hypertext applications, many other systems use metaphors in order to support structuring (e. g., many graphical based file system browsers.) They offer folders on which the user can drop documents. Other instances are card based document viewers (e. g., *Adobe Acrobat Reader*) which allow the display of pages of a document side by side in WYSIWYG mode, as they might be placed on a table.

The amount of content is increasing rapidly. One way to address this problem is to improve content engineering tools [25]; another is to support more informative structures without increasing cognitive overload. This is important especially for structures that are limited to spatial dimensions. Our approach focuses on the latter mentioned aspect, but goes beyond computer applications. A closer look shows that paper structures offer attributes and dependencies not found in any spatial structure supporting application. In this article, we compare structures created with paper to a selection of computer applications. We argue that the small additions with paper structures include information that improves finding and organizing. This will show that applications fail to leverage these advantages if they focus on a higher level of abstraction.

In Sect. 2, we relate our approach to others regarding how people use paper to organize their work. With this information, we will discuss paper-based metaphors on computers in Sect. 3. In Sect. 4, we will conclude and point to our future investigations.

## 2 Real World Structures

This section gives some idea of how real world structures are discussed in the field of computer science. We will point out how our approach compares to past efforts.

It is an interesting observation that there is only little related research work published. ("Despite the past importance of paper archives in office work, there are still relatively few studies of their nature and function"[1] [32].)

[1] In this context, [32] refers to [5, 16, 17, 19]. Other related work is [28, 15, 11].

One oft-cited article about how people structure their desks was written by Malone [19]. How do we distinguish our work from his? First, Malone applies a high level of abstraction to his structure description. Figure 1 shows two office maps that Malone was analyzing. All paper pile representations are right-angled. It can be assumed that this does not reflect the real world, especially not the right map, which is "filled with loosely stacked piles of mixed content" [19]. Malone does not focus in detail on how piles look like. Rotation is not considered.

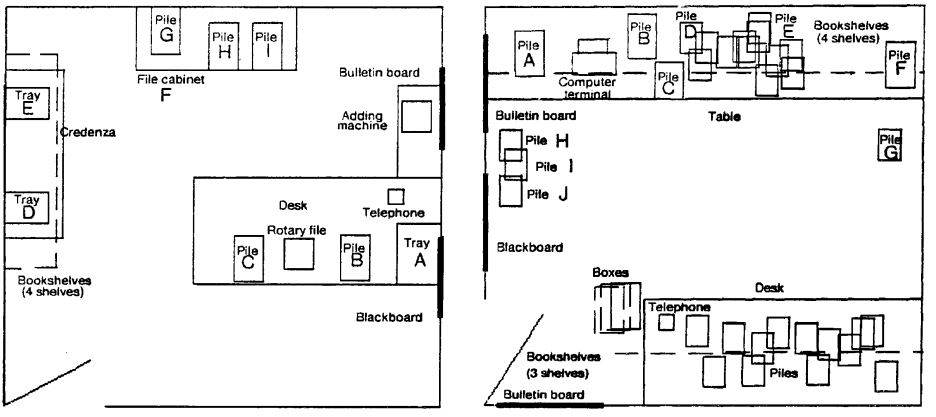



**Fig. 1.** Maps of Two Offices by Malone [19]

In comparison to that, we strongly focus on a more detailed description of the shape of piles and other structures as well as on additional aspects, such as binding mechanisms. With "binding mechanism" we describe the force which binds objects. Examples are depicted in Fig. 2.

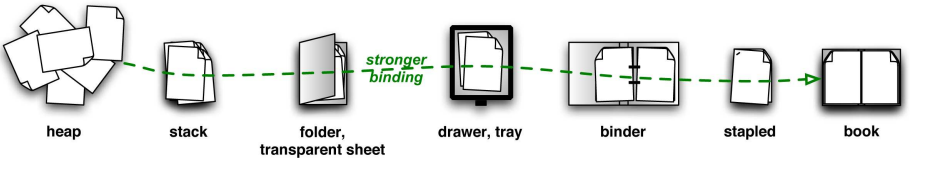



**Fig. 2.** Grouping Mechanisms, Sorted by Grouping Strength

Another difference can be recognized in terminology and level of details. Referring to [28], Malone's basic structure types are *file* and *pile*:

> "*[F]iles* are units where the elements (e. g., individual folders) are explicitly titled and arranged in some systematic order (e. g., alphabetical or chronological). In some cases, the groups themselves (e. g., entire file drawers) are also explicitly titled and systematically arranged; in other cases, they are not. ... In *piles*, on the other hand, the individual elements (papers, folders, etc.) are not necessarily titled, and they are not, in general, arranged in any particular order." [19]

We focus on the structure only, not on the "content". We do not distinguish between ordered and unordered according to its content. Malone's examples of an unordered bookshelf, which he calls a pile, and sorted shelves in a library, which he calls files, would be the same to us.

## 3 Paper-Based Metaphors on Computers

We will compare paper structures with three computer systems. Two are known as spatial hypertext applications: *Visual Knowledge Builder (VKB) 1.50* [12, 26]; and, *Tinderbox 2.2* [3, 9]. One is a diagram and chart application: *OmniGraffle Pro 3.1.2* [24]. All applications are tested on *Mac OS X 10.3*; *VKB* is implemented in *Java 1.3*, *Tinderbox* and *OmniGraffle* are native *Mac OS* applications. We see all three as applications used for spatially structuring of knowledge.

There is a survey on how students use *VKB* for magnetic poetry [26]. Figure 3 shows two pictures which are taken from that publication. We analyze them as an example to see differences of real world objects and *VKB*. We also analyze relevant features and limitations in *Tinderbox* and *OmniGraffle*.

### 3.1 Rotation and Sloppiness

As Fig. 3 depicts, all objects of the real world magnetic poetry show a certain rotation. Some yellow sticky notes seem to be rotated on purpose; at least the rotation angle is significantly different. Neither *VKB* nor *Tinderbox* can rotate objects.

*OmniGraffle* allows rotation any object, as shown in Fig. 4. Rotation can be applied using the mouse by pressing the command key. It allows 360 different rotation positions. The current angle is shown in a pop up window during the rotation process.

Even though *OmniGraffle* offers rotation objects, rotation must be performed explicitly. Most rotation on real world objects emerge while building a structure, as the magnetic poetry picture shows. On the other side, even intended rotation, as we assume for the yellow sticky notes shown in Fig. 3, cannot be taken over to *VKB*, because rotation is not supported.

Another observation is that most real world structures do not follow an exact grid, whereas all three selected application support that: *Tinderbox* has an invisible fixed grid of factor 0.125 of the default height of a note.[2] The grid cannot be switched off. *VKB* has a fixed grid which can be switched on and off. It is visible through dots as depicted in Fig. 5. The screenshot in Fig. 3 does not have the grid switched on.

*OmniGraffle* has a grid which size can be chosen arbitrarily and which can be switch on and off. As shown in Fig. 5, it has several additional visual features, e. g. if it should be dominant or how detailed the grid should be divided in smaller parts, etc.

[2] This information was given by Mark Bernstein, Chief Scientist of Eastgate Systems Inc., in a personal e-mail, dated 2004-05-06.

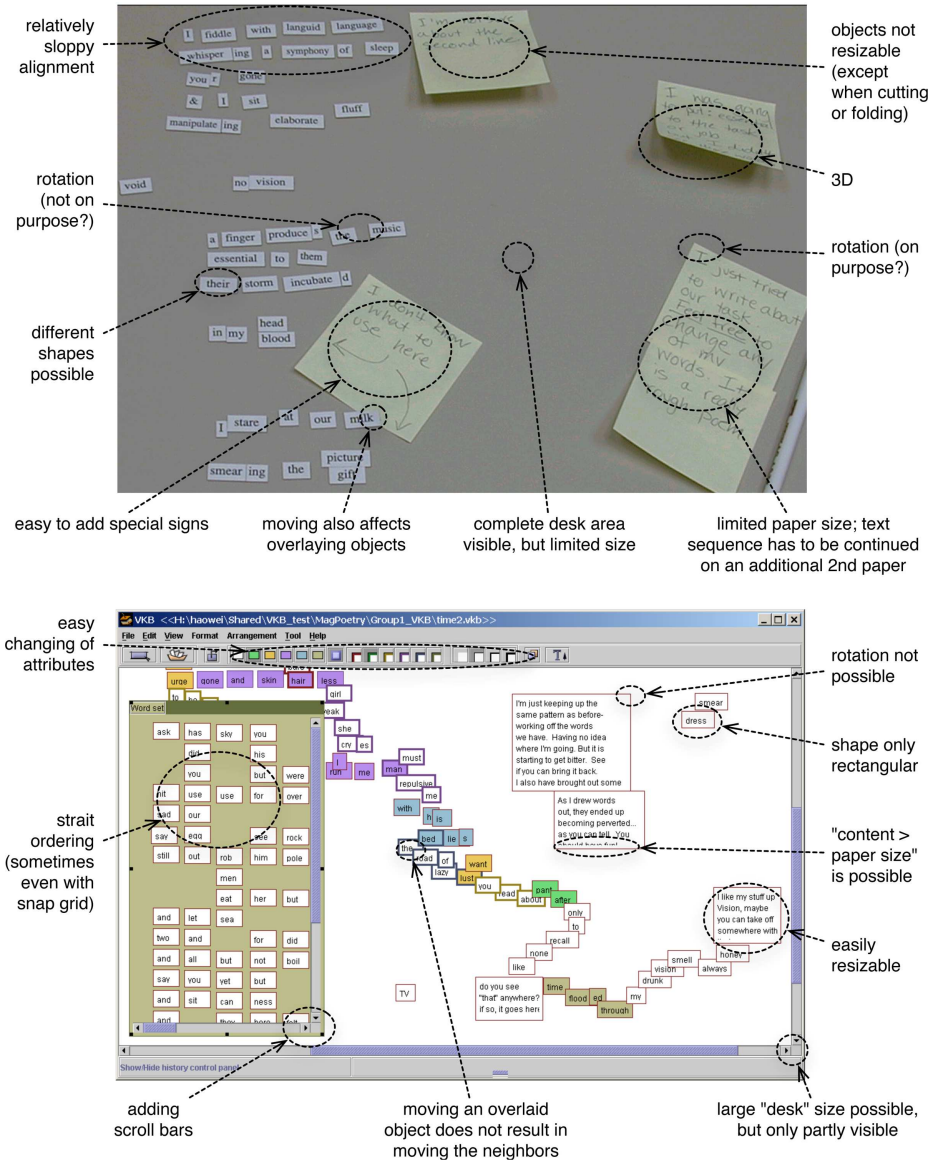



**Fig. 3.** Comparison of Structure – Real World and Spatial Hypertext Application (Pictures taken from [26] with kind permission of the first author)

Another aspect are alignment functions. *Tinderbox* does not offer any, but *VKB* and *OmniGraffle* do, as shown in Fig. 6. Those can be used to automatically align selected objects. Beside the standard alignment functions like horizontal or vertical alignment, *VKB* allows the arrangement of objects as stack as depicted. *OmniGraffle* has additional functions to align objects to the grid,

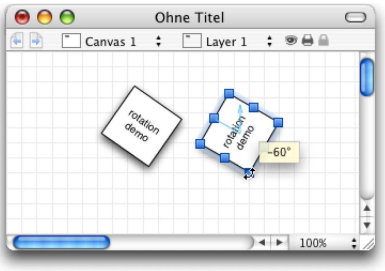



**Fig. 4.** Rotation in *OmniGraffle*

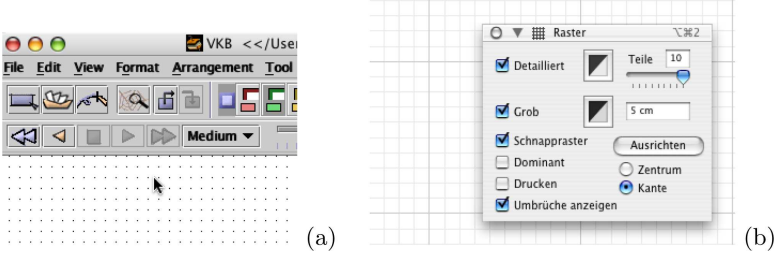



**Fig. 5.** Grid in *VKB* (a) and Grid and Grid Tool in *OmniGraffle* (b)

which can be seen in Fig. 5 as part of the grid inspector window. In the real world, such behavior exists partly. For example, loose paper on a desk can be transformed into a stack easily by moving the hand over the table and collecting the paper at once. However, the result will not look as perfectly aligned as it does in any of the computer application.

Rotation, whether being done on purpose or through emerging behavior and sloppiness, adds individual characteristics to spatial structures. The rotation of an object may express that this object is important; sloppy shaped piles may show that they are more often used, etc. For these reasons, we argue that rotation – done on purpose or done automatically by the system during structuring (e. g., when objects are moved) – helps the user to remember structures more easily and supports quick finding. If all documents show exactly the same direction, other aspects have

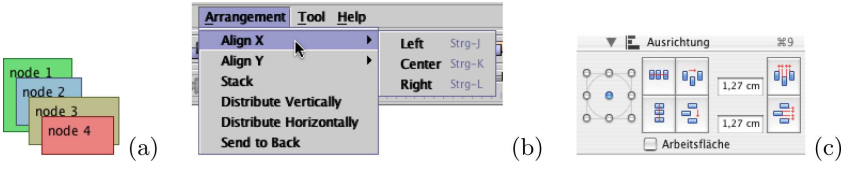



**Fig. 6.** As Stack Aligned Nodes in *VKB* (a) and Alignment Tools in *VKB* (b) and *OmniGraffle* (c)

to be perceived for finding (e. g., reading parts of the content.) It is likely that in many cases this takes longer than perceiving and processing rotation.

We expect that the user will rotate very important documents in order to find them or to be reminded more easily later. We also expect that people will not start to align documents precisely but accept the slight rotation done by the system based on the simulation of paper.

### 3.2 Shape, Size, and Collection Objects

Paper used in offices usually has a rectangular shape. *VKB* as well as *Tinderbox* follow this idea and provide only rectangular nodes. However, external graphics may be included that show shapes other than rectangular. *OmniGraffle* is the only application among the observed ones that allows the creation of arbitrary shapes. A stencil inspector allows to drag and drop any predefined shape onto the workspace.

Size is another important difference. The object size of the magnetic poetry shown in Fig. 3 is related to the word length. The height seems to be equal for all magnet objects. In most offices, the paper size used relates in most cases to standards and does usually not get changed (e. g., by cutting the paper.) Once a paper is cut, it is difficult to extend it again. All three observed computer applications, however, allow the resizing of objects without damage.

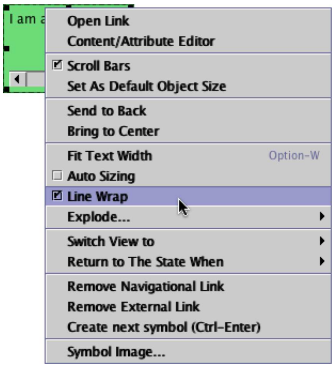



**Fig. 7.** *VKB*'s Context Menu for Nodes

Figure 3 shows two yellow sticky notes at the lower right corner of the real world picture that are put together. The reason is that the content at the first paper became too large. A second note was put to extend the first one. The picture below shows some equivalent notes in *VKB*. Also here, the text exceeds the visible size of the note. However, a closer look shows that the note acts like a window which displays the text only partly. The user is able to scroll the text with the text cursor. Also, resizing the note will cause more text to be visible. Additionally, *VKB* offers scroll bars, automatic sizing, and line wrap. These functions can be switched on and off individually for nodes at any time through the context menu shown in Fig. 7.

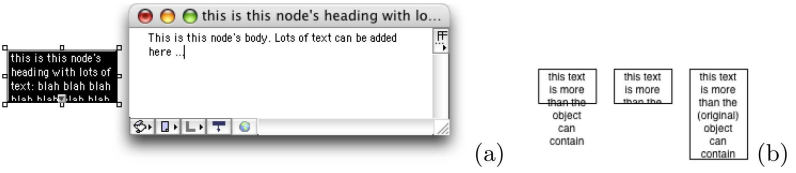



**Fig. 8.** Node with Heading in Map View and its Node Content in *Tinderbox* (a) and Nodes with Different Attributes in *OmniGraffle* (b)

*Tinderbox* displays as much text as possible on a node and hides the rest. Only the node heading is represented on the node. The user cannot scroll headings as with *VKB*. A heading can be edited through a dialog window. Also, a node can contain text beside the heading, but it will not be displayed within the spatial representation; it can be displayed in a separate window or be exported in another format (e. g., HTML.) The spatial representation of a node with a long heading and its content window is shown in Fig. 8.

*OmniGraffle* offers three modes for handling texts that are too long. Fig. 8 gives an example for each mode. The left one continues printing the text over the node's border. The second node just cuts the text. The last one adjusts the node's height automatically according to the space the text needs.

A collection object follows the metaphor of a drawer or box in which the user can put things to collect them. *OmniGraffle* does not support them. One example of a *VKB* collection can be seen in Fig. 3. It looks like an additional window. It has a different background and a title. The scroll bars indicate that the collection space is larger than the visible area; however they also can be switched off. *VKB* has functions to scale the content of collection objects. In *VKB*, collections are different objects than nodes.

In *Tinderbox* any node can also serve as a collection object. The first picture of Fig. 10 depicts a node which contains other nodes. Similar to *VKB*, it contains a single line with the node's title. However, it shows a content preview at a fixed factor. It's size can be changed at any time by changing the size of the collection object. The last picture of Fig. 10 shows the inside of the collection object. The size of the containing node is indicated as a frame on the background. This is also the area which is visible at the collection node itself.

In the real world, people use drawers, boxes, etc. to collect objects. Such collection facilities have specific borders that are basically the same, inside and outside. This limitation forces reorganization whenever a container becomes too small (e. g., by starting a second container.) In comparison, *VKB* or *Tinderbox* collections can consist of a larger space inside than its shown size at the outside. A computer application can simulate the real world behavior by limiting the possible number of objects for collection objects or binding mechanisms.

Text on paper is limited to the paper's size. This has the advantage that large texts can be found by approximating the amount of paper. This can be simulated in a computer by allowing only fixed size documents without scroll bar, writing beyond its border, or the possibility resizing a document. Whenever

there is more text than can be put on one page, another page has to be created. These pages must be movable individually if no binding mechanism is applied.

In addition to the simulation of binding mechanisms such as binders, stapled documents, or loose piles, we claim that fixed size documents statistically significantly decrease the time for finding due to the following reasons:

1. Objects can be found by perceiving the number of documents they contain. For example, a 500 pages PDF file will have 500 individual pages displayed.[3]
2. The user is forced to reorganize structures if they become too large. For example, in the real world, a binder can hold only a certain number of sheets. If this number has to be extended, another binder must be created. This means that it is less likely that one level of structure becomes too complex, because it has to be reorganized before it becomes too large. Additional levels are pushed into an existing structure. Less complex structure levels will help to find information.
3. Since there is a fixed size for all documents, there is no document that is harder to find than others because its small size.[4]

### 3.3 Zooming

Zooming can be seen as jumping between structure levels. This is done very smoothly in the real world. As an example imagine a secretary who needs to read about a certain meeting. The paper is placed in a binder in another office. The first structure level is how the offices relate to each other spatially. When she enters the right office, she dives into the next structure level, the spatial ordering of the furniture. She continues to zoom into the bookshelf and resolves the structure there until she finds the right binder. The next structure level is the inside of this particular binder, which she has to process. She will find the paper about the meeting. The secretary is finally looking for the right paragraph and finally she moves to the structure level of sentences and below while she is reading.

As this example shows, there are may different structure levels. A person usually can switch between them very smoothly, in many cases even without noticing. In some cases, it also can be observed that someone switches back and forth constantly, e. g. when several papers inside different binders have to be found and read.

Compared to this, the zooming features of current computer applications are poor. The observed ones allow basic zooming. *VKB* offers three different zoom

[3] This includes 3D or semi-3D representation (e. g., darkness of shadow or semi-3D representation of the height of a pile, in order to show how many pages are on a heap, stack, etc.)

[4] We are aware of the fact that different sizes can be seen as an attribute that helps to distinguish between documents and therefore helps to find objects faster. However, also the empty space of a fixed size document with little text on it versus a fully covered page can be seen as an attribute which does this. Therefore, having different document sizes versus different border sizes does basically the same in this respect.

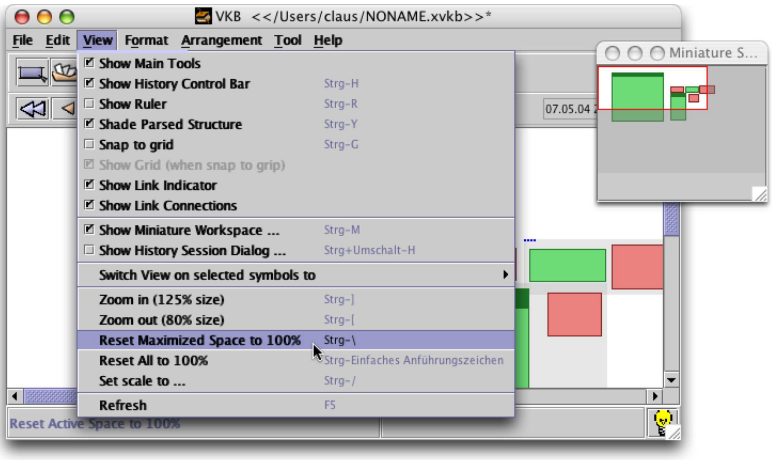



**Fig. 9.** Zoom Functions and Floating Miniature Workspace Window in *VKB*

scaling directly by menu, as shown in Fig. 9: 125 %, 80 %, and 100 %. They apply to the current scaling. The scale factor can also be set arbitrarily via a dialog window. *Tinderbox* offers a pop up list at the spatial structure windows with zoom levels from −4 to 4. The middle position is called "Normal". Similar to *Tinderbox*, *OmniGraffle* has a zoom pop up menu. It shows basic zoom scales in percentage which can be selected by one click. The menu also allows one to enter an arbitrary zoom factor manually. In addition to this, *OmniGraffle* maps the mouse wheel to zoom in and out when holding the command key. However, a large number of objects slow down smooth zooming significantly.

As shown in Fig. 9, *VKB* offers a floating miniature workspace window which gives an overview of the complete workspace. The user can imitate quick zooming by switching between the main window and the miniature. The red border shows the visible area at the main window. It can be moved directly using the mouse.

*OmniGraffle Pro* allows an arbitrary number of views of the same document represented in individual windows. This can be used to have one window scaled 100 % and another one 10 %. Changes can be made to both and are seen immediately in all views. This allows switching smoothly between an overview and a closer look. However, there is evidence that a separate overview causes a slower performance in finding [14].

There is another kind of zooming in *Tinderbox*. It is an animation of moving into a node. Fig. 10 shows a sequence of pictures. The duration and the number of visible zoom steps depend on the window size, the processor speed, and the number of items at the destination space. The depicted example had 5 different zoom steps visible. The duration of the whole process was approximately 0.3 sec.

Even though this visualization is mostly smooth, it cannot be stopped in between steps. It shows a node as start state and the node's space as end state.

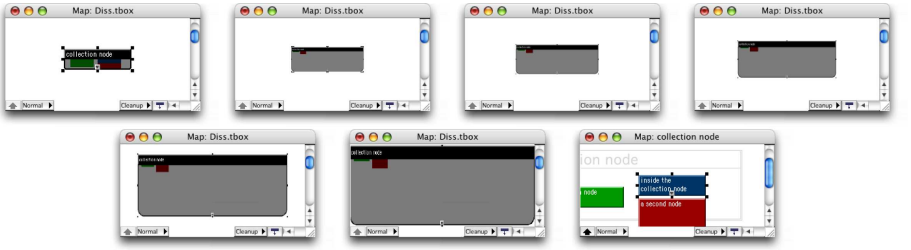



**Fig. 10.** Sequence of Zooming into an Object in *Tinderbox* (Five Different Zoom Steps Visible in Total, Duration Approximately 0.3 Sec)

It is not possible to stop in between or have a node's content side by side to objects in other structure levels, as may be the case at the real world.

These applications' zooming functions are not satisfying compared to the real world. They require an additional request of an exact number of how much scaling the user wants to apply. Closest to the real world is *OmniGraffle* when using the mouse wheel for zooming.

Inspired by the real world, we believe in easy applicable and smooth zooming with no predefined zoom steps for spatial structuring tasks. Basically, the user should be able to use any zoom factor inside a reasonable scale. We claim that smooth zooming will decrease the time for finding documents or structures statistically significantly. Based on the visible change without breaks in between, the user has a better orientation of where a zoom call leads to spatially and he/she is able to increase or decrease the visible workspace area very quickly. Because of this, the time for reorganizing objects will be also statistically significant faster.

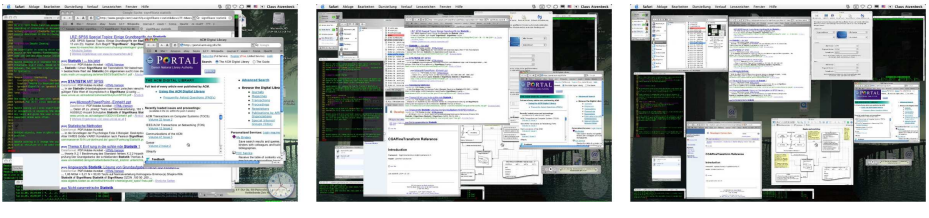


**Fig. 11.** Sequence of Resizing and Repositioning of Windows with *Exposé* on *Mac OS X* (Duration Approximately 0.23 sec)

One example can be experienced using *Exposé*, a built-in function of *Mac OS X* since version 10.3. It reduces the size of windows and moves them side by side so that the user can select the desired window very quickly. Fig. 11 shows some screenshots taken during this transformation, which lasts approximately 0.23 sec. in total. With shift key pressed, the speed is reduced: The same action takes over 2.5 sec. However, it can be assumed that the user gets a better understanding of where things go. This is also supported by the fact that the contents

of visible windows do not freeze. For example, running movies don't stop when *Exposé* is activated.

Another related approach is *Piccolo*, an open source Java library which allows the building of two dimensional zoomable user interfaces. *Piccolo* is the successor of *Jazz* [2]. One aim is to support smooth and continuous zooming – so called "semantic zooming" – for a large number of arbitrary objects. There is evidence that zooming of workspaces with many objects improves the recall compared to the use of scrollbars [6].

A similar survey compares the recall of a zoomable user interface with an overview of the shown workspace to one without [14]. It has been shown that the finding task was solved faster without an overview, even though 80 % of the subjects preferred to have one.

The mentioned examples support our hypothesis that smooth zooming helps to find objects on a 2D workspace faster compared to fixed zoom steps or the use of widgets like scrollbars or additional overviews. Additionally, we expect that the user's subjective satisfaction will be increased and that the type of created structure will change mainly from classification to spatial structures.

### 3.4 Other Observations

Other observations we made cover 3D, simulation of gravity and friction, desk size, and interaction.

We have found that real world behaviors, like 3D, friction and gravity, are used to relate objects. For example, a small paper on top of a large one will also be moved when the large one is being moved. None of the observed computer application allows this, except *OmniGraffle* when grouping is switched on. However, this has to be done explicitly and is only a poor simulation of real world forces.

Desk size is another difference. For example, according to our tests, *OmniGraffle* allows a maximum document size of 3527.8 km × 3527.8 km, which is an area larger than the USA or Canada. In the real world, there is no desk that can be even nearly as big. The user is forced to reorganize according to the given desk limitation. This has influence in how the user structures information.

Objects in computer applications are mostly moved differently compared to objects in the real world. The most common input devices are keyboard and mouse. A direct touch on the visible object does not have any result on ordinary screens. Touch screens close this gap to some extent. They allow direct manipulation at the same spot at which the object is displayed. Also writing on the screen directly becomes possible. Digital output devices in general are small compared to real world structures which mostly take much more space. Large screens or projections on tables (e. g. described by [1]) offer much more space for representing digital documents.

## 4 Conclusion and Future Work

Applications which can be used for spatial knowledge structuring are based on the metaphor of cards and trays. They implement an abstract level of the real world. Details like emerging rotation or sloppy alignment are left out and in many cases even seen as disturbing. Smooth zooming or limitations like fixed size objects are not implemented.

In our paper we pointed out that it is likely that those details at the real world carry additional information and possibilities of interactions for spatial structuring in knowledge work. It is likely that this information can be processed in parallel with no additional cost, since a person usually does not process or reflect it consciously. In order to prove this, we will implement a test environment which enables both kinds of rotation, emerging and purposeful, fixed size documents with binding mechanisms, and smooth zooming. We will test the speed for organizing and recalling information spatially with and without those features switched on. Structural computing systems [33] offer a good environment to implement our application, including structure services for rich spatial structures and structure awareness.[5]

We follow a different paradigm in graphical user interfaces for spatial structuring in knowledge work: The detailed implementation of metaphors. This would contradict with systems nowadays, which seem to extend the capabilities of the real world and cut down imprecise behavior. They cut down additional information, as we have shown, which is helpful for the user.

Paper is a medium with thousands years of history. People are very used to it and have lots of experience in organizing it. Lessons learned from observing paper structures in the real world, our new approach in computer science focuses on rich structures to help "augmenting human intellect" [10], as Engelbart was advertising for already in 1962.

## References

1. M. S. D. Ashdown. Personal projected displays. Technical Report 585, University of Cambridge Computer Laboratory, 3 2004.
2. B. B. Bederson, J. Meyer, and L. Good. Jazz: an extensible zoomable user interface graphics toolkit in java. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, pages 171–180. ACM Press, 2000.
3. M. Bernstein. Collage, composites, construction. In *Proceedings of the 14th ACM Conference on Hypertext and Hypermedia*, pages 122–123. ACM Press, 2003.
4. V. Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 7 1945.

[5] An important aspect is the implementation of data, structure, and behavior as different views [23, 30] for rich structures and binding mechanisms, which affects especially the structure services. A recent conference contribution shows an example of how this paradigm can be applied to an application [29].

5. I. Cole. Human aspects of office filing: implications for the electronic office. In *Proceedings of the 26th Annual Meeting of the Human Factors Society*, pages 59–63, 1982.
6. T. T. A. Combs and B. B. Bederson. Does zooming improve image browsing? In *Proceedings of the 4th ACM International Conference on Digital Libraries*, pages 130–137. ACM Press, 1999.
7. 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.
8. 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.
9. Eastgate Systems. *Tinderbox$^{TM}$ for Macintosh v. 2.2. User's Manual & Reference.* Eastgate Systems, 2004.
10. D. C. Engelbart. Augmenting human intellect: A conceptual framework. Summary Report AFOSR-3233, Standford Research Institute, 10 1962.
11. D. Frohlich and M. Perry. The paperful office paradox. Technical Report HPL-94-20, Hewlett-Packard Laboratories, 3 1994.
12. K. Gupton and F. Shipman. *Visual Knowledge Builder version 0.70. The user's manual.* Center for the Study of Digital Libraries, Texas A&M University, 2000.
13. F. Halasz and M. Schwartz. The dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, 2 1994.
14. K. Hornbæk, B. B. Bederson, and C. Plaisant. Navigation patterns and usability of zoomable user interfaces with and without an overview. *ACM Trans. Comput.-Hum. Interact.*, 9(4):362–389, 2002.
15. F. Khan. A survey of note-taking practices. Technical Report HPL-93-107, Hewlett-Packard Laboratories, 12 1994.
16. A. Kidd. The marks are on the knowledge worker. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 186–191. ACM Press, 1994.
17. M. Lansdale. The psychology of personal information management. *Applied Ergonomics*, 19:55–66, 3 1988.
18. K. Lyon and P. J. Nürnberg. Applying information visualisation techniques to spatial hypertext tools. Proceedings of the MIS'04 Symposium, Salzburg, Austria (in print), 2004.
19. T. W. Malone. How do people organize their desks? implications for the design of office information systems. *ACM Trans. Inf. Syst.*, 1(1):99–112, 1983.
20. 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 3rd Annual ACM Conference on Hypertext*, pages 261–275. ACM, ACM Press, 1991.
21. 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.
22. 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.
23. P. J. Nürnberg, U. K. Wiil, and D. L. Hicks. A grand unified theory for structural computing. In D. L. Hicks, editor, *Metainformatics. International Symposium (MIS'03), Graz, Austria*, volume 3002 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2004.

24. Omni Group. *OmniGraffle 3*, 2003.
25. S. Reich. Content engineering: Bridging the gap between content creation and consumption. a position statement for mis 04. Proceedings of the MIS'04 Symposium, Salzburg, Austria (in print), 2004.
26. F. Shipman, R. Airhart, H. Hsieh, P. Maloor, J. M. Moore, and D. Shah. Visual and spatial communication and task organization using the visual knowledge builder. In *Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work*, pages 260–269. ACM Press, 2001.
27. R. Simpson, A. Renear, E. Mylonas, and A. van Dam. 50 years after "as we may think": the brown/mit vannevar bush symposium. *Interactions*, 3(2):47–67, 1996.
28. D. Tsichritzis. Form management. *Communications of the ACM*, 25(7):453–478, 1982.
29. A. Ulbrich and K. Tochtermann. Applying structural computing paradigms to domain analysis by example of elearning in higher education. Proceedings of the MIS'04 Symposium, Salzburg, Austria (in print), 2004.
30. M. Vaitis, M. Tzagarakis, K. Grivas, and E. Chrysochoos. Some notes on behavior in structural computing. In D. L. Hicks, editor, *Metainformatics. International Symposium (MIS'03), Graz, Austria*, volume 3002 of *Lecture Notes in Computer Science*, pages 143–149. Springer, 2004.
31. H. Van Dyke Parunak. Don't link me in: set based hypermedia for taxonomic reasoning. In *Proceedings of the 3rd Annual ACM Conference on Hypertext*, pages 233–242. ACM Press, 1991.
32. S. Whittaker and J. Hirschberg. The character, value, and management of personal paper archives. *ACM Trans. Comput.-Hum. Interact.*, 8(2):150–170, 2001.
33. U. K. Wiil, D. L. Hicks, and P. J. Nürnberg. An agenda for structural computing research. Proceedings of the MIS'04 Symposium, Salzburg, Austria (in print), 2004.

# Towards a Generic Building Block for Component-Based Open Hypermedia Systems

Omer Ishag Eldai, Peter J. Nürnberg, Uffe K. Wiil, and David L. Hicks

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

**Abstract.** In this paper, we propose a framework for component-based open hypermedia systems that provides developers with ready-made and extensible communication and distribution facilities. We begin by analyzing the requirements for such a framework through an examination of existing systems. We then describe our framework, and tie this back to related work in the metainformatics field.

## 1 Introduction

Due to current trends in design and development of component-based open hypermedia systems (CB-OHSs), these systems are becoming more modular, flexible, and open. In that direction, we are trying to further advance the state of art by proposing a generic building block for system development that improves architectures, making them more flexible, and hence making systems easier to build and easier to update. To achieve this, we have done a survey to identify aspects and issues in building systems. We used this survey to inform the design and implementation of a proposed building block for CB-OHSs to address all the aspects identified in our survey.

Distribution and interoperability have become very important in today CB-OHSs, since users and services are massively distributed. Hence system developers have to consider these aspects in early system design stages. Otherwise, it will be very difficult to support these features.

The results that we have drawn from the survey of some prominent CB-OHSs showed that most existing systems do not satisfy users and developers needs.

An aspect is a desirable feature. This feature when introduced into the system increases the complexity or effort needed in system design and development. We named them as aspects, because we believe that introduction of any of these aspects to the system increases the complexity of its design and development. We have divided these aspects into two categories according to the way it is addressed (direct/indirect) by our proposed building block.

The rest of the paper is organized into the following parts: Sect. 2 provides a historical introduction to the research area of CB-OHSs. A brief definition of the

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 66–84, 2004.

set of the aspects that we believe to be considered in designing CB-OHSs is given in Sect. 3. In Sect. 4, we present the analysis framework used to perform the survey, and the scenarios supporting the importance of those aspects. Section 5 shows the important results that we have drawn from the survey. The proposed building block, its advantages, its improvements on aspects, and its support for users and developers of CB-OHSs are detailed in Sect. 6. Section 7 describes the relations between the aspects. Related work including structural computing and recent trends in design of CB-OHSs are shown in Sect. 8. Finally, Sect. 9 includes our conclusions and future work.

## 2 Hypermedia Systems History

The history of hypermedia systems can be characterized by a set of revolutions. In each revolution, there is a group of hypermedia systems that have been developed. These systems share common criteria and features and mostly address the same set of issues. In this history review, we will describe the revolution and then highlight common features or trends of the systems developed at that time. We will also provide names of some of the prominent systems that represent that era.

### 2.1 Bush Era (1945–1987)

This era began in 1945 when the hypermedia pioneer V. Bush published his ideas about MEMEX [1]. Based on Bush's ideas, a group of systems were developed such as KMS [2], NoteCards [3], and Intermedia [4]. The systems developed during this period were monolithic systems[5].

### 2.2 Dexter Era (1988–1996)

After the Dexter Model was introduced to the hypermedia research community [6, 7, 8], a group of systems implemented those ideas. As a result many advances occurred in hypermedia systems development. Examples of these advances include the client-server architecture, openness of architectures, integration tools, and development tools. Many systems were developed during this era such as DHM [9, 10], Microcosm [11], HOSS [12], HyperDisco [13, 14], and Chimera [15, 16]. This era is characterized by client-server-based and open hypermedia systems.

### 2.3 OHSWG Era (1997–2003)

The OHSWG era is characterized with its benefit of the efforts performed to achieve standardization and interoperability. Systems developed at this era adopted the architecture openness, component-based approach, and the structural computing philosophy introduced in HOSS [12]. Current systems that implemented these advances include FOHM [17, 18], Callimachus [19], Construct [20, 21, 22], and Themis [23, 24]. The main common features between these

systems are the adoption of the component-based approach (CB-OHS) and structural computing philosophy and the generalized data model such as in FOHM.

### 2.4 New Era (2004–)

This era represent the systems that will come out as a result of the active CB-OHSs research community. We believe that there will be new versions of exiting systems as well as newly developed systems will appear in the near future. We think that the main direction of these systems will be more work on structural computing, openness, and implementation of these systems in new structural domains.

## 3 Aspects

The set of the aspects to be considered when designing an OHS are:

- *Structure awareness.* This refers to the architectural level where the system is aware about structures (Interface/Display, middleware, backend, and operating system level).
- *Distribution.* This aspect involves system support for different configurations, the support for different hardware and software platforms, and the allocation of the system itself in one or several locations, and its services within the environment.
- *Architecture openness.* This aspect describes the capability of the systems architectural model to support modularization and scalability.
- *Open data abstraction.* This addresses the capability of a system to support and define different/new data types, and facilitate the integration of applications to the system
- *Open structure abstraction.* This is the capability of the system to support different sets of structural abstractions, and structural services and to extend, update the set of available structure abstractions and structural services.
- *Open services levels.* This refers to the ability of the system to provide different, meaningfully independent levels of services without forcing the user to subscribe to undesired level of service.
- *Interoperability.* This is the ability of the OHS components to know each other, communicate with each other and the ability of the system to communicate with the integrated applications, WWW, and other OHSs.
- *Maturity.* This aspect concerns whether a system design and development are based on existing standards.

## 4 Scenario Descriptions and Analysis

In each subsection below, we provide two parts: a scenario description; and, as analysis. In the scenario description section, we provide a short story to explain

the needs of users of hypermedia authoring environments. In the analysis section, we state the goals of the scenario, state the participating characters needs and the problem, analyze the requirements on developers and the our proposed building block (BB), and conclude with the solution provided by the building block for the stated problem.

The characters in this scenario (A, B and C) are working at Elegant Software Company. They have performed the analysis, design and coding for a stock control subsystem. Now they want to collaboratively document the subsystem that they have developed. The scenario has the following characteristics:

- Multiple users (co-editors) are located at different sites.
- Each editor uses a different hypermedia-authoring environment.
- The editors need to share some of the documents.
- There is a need to support both asynchronous as well as synchronous collaboration.
- Users are free to use different service levels.

The scenarios describe a sample task (co-editing) between a group of editors and analyzes the requirements on cooperative open hypermedia systems to support them.

In the scenarios, author A and author B are working on a group of shared hyperdocuments that consists of a network of objects and structures (nodes and links) using the HMSs that they acquire in the initial scenario.

### 4.1 Scenario 1: Support for Distribution

**Scenario Description.** A is a software engineer at Elegant Software Company. He has decided to explore the use of the hypertext to support a documentation task has been assigned. He installs a free HMS on his group computer (IBM compatible PC, with Windows OS) and starts creating links to and from the documentation and code using HMS-A.

When A and B meet to discuss the status of this task, A tells B that he is using HMS-A to create links/structures. B finds the idea of using a HMS very interesting and hence, he decides to install and use it on his own laptop (Mac). A and B would then be able to import the documentation from one machine to another and exchange other analysis, design, and code documents. When A attempts to install the HMS on Bs machine he finds that the HMS does not work properly. A is quite sure that he followed the same steps he used when installing HMS-A on his own PC. After several unsuccessful attempts he decides to contact the developer of HMS-A. He writes an e-mail message to the developer of HMS-A asking for help. He is informed that the system cannot run on the Macintosh platform.

**Scenario Analysis**

*Goals.*

- *Support for multiple platforms:* Support for system availability to run on different platforms (hardware and software).

- *Support for availability on different configurations:* The possibility to configure the system to run on LAN, WAN and the WWW.
- *Support for standardization:* Support of the previous two points means support for standardization (development of systems that can run on different platforms), and different configurations (this may be facilitated through development of scalable/sizable systems).

*Problem statement.* It is obvious that the hypermedia system (HMS-A) does not support different platforms and network configurations as required by the characters.

*Requirements for developers.* Developers can deal with these issues by creating platform independent systems. Many techniques can contribute to the achievement of this goal, including the implementation language, development of reusable components and, adoption of standards. In addition, a developer can choose to develop a single version of the system that is platform independent or develop different versions of the system for different platforms and configurations.

### 4.2 Scenario 2: Support for Open Data Abstraction

**Scenario Description.** Author B searched the Web for an alternative HMS. He finds and installs another hypermedia system that supports the Mac platform. B starts creating links between his documents and code. He then contacts A and they decide to meet to exchange documents. During their meeting, they discover that they are using different hypermedia vocabulary terms and so it is difficult for them to understand each other. Moreover, they are using different editors and discover that they cannot copy documents from one system to another due to the use of different data formats.

**Scenario Analysis**

*Goals.* The goals of this scenario are to demonstrate the need of users of different open hypermedia systems to exchange hyperdocuments and data without consideration for the authoring applications that they use. The second aim is that users expect to be able to communicate their knowledge with other colleagues using different systems, and to be able to use different HMSs for creating structures and links.

*Problem statement.* Users A and B are unable to exchange ideas and knowledge about the tasks they have performed because the two hypermedia systems use different conceptual frameworks and hence lack a shared vocabulary. The characters in the scenario are unaware of the data formats supported by each of the hypermedia systems, and hence they think they can exchange hyperdocuments.

*Requirements for developers.* Developers must create hypermedia systems that support different data formats, either directly or indirectly. They also have to use well-known (standard) terms in their systems, so that a user of one system can more easily use other systems (standardization support).

### 4.3 Scenario 3: Support for Distribution and Maturity

**Scenario Description.** A and B decide to look for a solution to the problems in the first two scenarios. A suggests that they can install HMS-A on their companys application server; they can then access the system from their offices or at home. They test the system on the server and it works fine. They try to access the system from their desktops concurrently and fail. A message to the developer of HMS-A reveals that the system does not support multiple users and does not support distributed access across a network.

**Scenario Analysis**

*Goals.*

- *Support for standardization in OHS design and development:* Support for using standard design methods (CB technology, modules, etc.), and implementation methods, selection of standard component framework technologies for communications purposes (RMI, CORBA, TCP/IP, SOAP, etc.)
- *Support for extensibility in OHS design and development:* Support for development of flexible, extensible systems architectures. This includes the ability to support the introduction of new components to a system and the deletion of existing components while the system is running. It also includes the introduction of extensible classes/components.
- *Support for distribution and sizing/scaling:* Support for the execution, reuse, and distribution of the systems components (As well as the whole system), services, clients/applications, and users over different configurations including single machine, LAN, WAN, and the WWW.

*Problem statement.* In practice, users may require systems with flexible and extensible architectures that allow them to either add a new service to a system or remove/delete an existing service without halting the system. They may need to modify an existing service or develop a new one using the templates of the running system. Many systems are developed without this support, however. The second issue is that many systems were developed to run on a few configurations; these systems do not allow scale well in WAN settings.

*Requirements for developers.* Developers must: design systems and services that can be distributed in many different network configurations; provide systems that can be upsized and down sized on the fly; supply users with extensible classes/components; and, support multiple users. They must also support distribution.

### 4.4 Scenario 4: Support for Architecture Openness

**Scenario Description.** The developer of HMS-A recommends a new version of the system recently released. The new version supports a LAN-based configuration as well as different users. Now A and B can use the system as desired. However, they would also like to use the system in a WAN-based configuration and they would like to use their favorite text editors, which are not currently supported by HMS-A. They decide to make some extensions to HMS-A so that it can be used across a WAN and support their favorite text editors. Unfortunately, they discover that the system is not extensible.

**Scenario Analysis**

*Goals.*

- *Support for application integration:* Support for application integration either as a part of the system environment (full integration) or be able to launch of those application that do not support full integration (partial integration).
- *Support for maturity:* Support for maturity through the standardization of the analysis, design, and code documentation, etc.

*Problem statement.* In a hypermedia environment, users prefer to keep their legacy applications and integrate popular editor into the environment instead of learning how to use a new editor. To satisfy these requirements, developers have to support the integration of these applications in a straightforward manner.

*Requirements for developers.* Developers must design open architectures to support different levels of application integration, and provide users with some means (documentation, integration tools, etc.) to perform application integration (support of maturity).

### 4.5 Scenario 5: Support for Open Service Levels

**Scenario Description.** Now all the software development staff at Elegant Software Company are using HMS-A. One day, A receives an error message when trying to open a design document. He discovers that another member of the group was already editing that document. This identifies a need to edit documents synchronously so that users can finish their work on time. They decide to develop a new collaboration service and integrate it into the system. The collaboration service allows two or more authors to edit the same document at the same time. In addition, they want the service to flexibly allow users to switch from a single user editing mode to a multi-user collaborative editing mode and back.

**Scenario Analysis**

*Goals.*

- *Support for open service levels.* Support for open service levels includes the ability of a service or system to switch from one level of service to another level without needing to start a new service. Suppose that an author editing a document in asynchronous mode, and then another authorized author starts using the service to edit the same document. The service should switch up to synchronous level so that both authors can co-edit the document without needing to start a different service. The same functionality is applicable if two authors are editing the same document simultaneously and one of them exits the service or decides to edit another document. The other author should be automatically switched to an asynchronous level.
- *Support for component interoperability.* Support for component interoperability can facilitate and support the pervious requirement explained in this scenario, and the possibility of development of only one component to be shared between many different components. A good example for such components could be a communication component, storage component and naming and allocation component.
- *Support for component-based approach:* Support for component-based approach will minimize and ease the efforts needed to modify/add component.

*Problem statement.* In performing their daily tasks and while using services, users would prefer to be switched automatically to the level of the service they need. This requirement forces the need for service interoperability and hence developers need to develop services in a CB manner with explicit support for different levels of service.

*Requirements for developers.* Developers must develop services/systems as components that can communicate and understand each other messages, and allow for multiple service levels (for those service that can be leveled e.g., collaboration, an taxo-spatial service, gaming service, etc.).

### 4.6 Scenario 6: Support for Interoperability

**Scenario Description.** One year later, Elegant Company decides to merge with Software House Company (SHC) which has many offices distributed over the world. SHC is already using another HMS (HMS-C). After the merge, A tries to create a link from his code to the analysis made by team member C, who is using HMS-C, and discovers that it is not possible, because the two systems use different communication protocols and can not exchange messages. The department head decides to extend HMS-C to be able to communicate with HMS-A through the addition of a new module to HMS-C.

**Scenario Analysis**

*Goals.*

- *Support for components and systems interoperability:* The support for systems interoperability includes the ability of the different hypermedia systems to communicate and understand each other messages. This can be either between whole systems or between services of the different systems. Achievement of this requirement will help users to exchange and share hyperdocuments as well as support collaboration.
- *Support of different communication protocols:* Support for different communications protocols includes support of well-known existing components framework technologies such as CORBA IIOP, Java RMI, SOAP and support of TCP/IP. Systems that comply with communication protocols standardization will support some of the famous common protocols, and accordingly, they will be able to communicate with each other.

*Problem statement.* Users using different hypermedia systems environments may need to interact with objects and documents maintained by other hypermedia systems. The kind of navigation needed may iclude browsing, and storage references to remote objects or documents. Thus, different hypermedia environments should support these interoperability requirements. The interoperability may be between services belonging to the same environment or between different environments.

*Requirements for developers.* Developers must design systems that support both components and systems interoperability (horizontal and vertical interoperability).

### 4.7 Scenario 7: Support for Structure Awareness

**Scenario Description.** While writing documentation, A and B try to make links to the arrows and other symbols representing the systems data flow diagrams and architecture diagrams. However, they find that HMS-A and HMS-C support linking only to the whole diagram and not parts within it. Hence, they decide to integrate an application that supports the creation and storage of the design and analysis objects (arrows, triangles, rectangles, etc.) as independent objects that can be treated separately and which also can be added to groups (composites).

**Scenario Analysis**

*Goals.*

- *Support for structure awareness.*
- *Support of structures at different levels:* including application/user interface, middleware, backend, and operating system.
- *Support for creation of structures between displayed objects.*
- *Support for creation of relations and links between displayed objects.*

*Problem statement.* A client of hypermedia environments (applications, browsers, users) needs to be able to access and display retrieved objects. If this is not possible, it may just start a browser or hypermedia system that maintains the required objects on the users workstation. Moreover, clients may require creating relations and links between the displayed/retrieved objects and hence storing the references to these structures either remotely or locally.

*Requirements for developers.* Developers must design a structure aware storage service.

### 4.8 Scenario 8: Support for Open Structure Abstraction

**Scenario Description.** At a department meeting, As department head suggests that a web page be created for the department; it can be used to house As and Bs documentation on-line. When the Web group starts developing the site they discover that taxonomic structures are the best way to structure the documentation and the analysis/design documents. Unfortunately, they discover that their OHSs do not support taxonomic structures.

**Scenario Analysis**

*Goals.*

- *Support for open structure Abstraction:* Support for open structure abstraction includes the support for different sets of structural abstractions, structural services and the ability to extend and update the set of available structure abstractions and structural services. The open structural abstraction level of a system is directly related to the number and types of supported structure abstraction in the specific system, as well as common structure abstractions (node, link, anchor, taxon, and composite) and structural services (navigational, spatial, taxonomical, argumentation).
- *Support for creation of structures (relation, link) between objects:* The system structural model support for an open set of structure types (link, anchor, node, taxon, support positions, and evidence structure abstraction).

*Problem statement.* A client of hypermedia environments (applications, browsers, users) needs to be able to navigate and organize structures in different ways. This requirement can be supported by the ability of the environment to allow creation and navigation of existing/displayed objects. Another factor is the structural data model support for an open set of structural abstractions. Many of the existing hypermedia system environments support only a limited number of structural abstractions and structural services.

*Requirements for developers.* Developers must design an open structural model to support an open set of structural abstractions and structural services, and design systems that support the creation of structural relations and link between objects.

## 5 Survey Results

We have surveyed some of the prominent OHSs including: DHM; Microcosm; Chimera; HOSS; HyperDisco; and, Construct. We designed an analysis framework to carry out the survey. The analysis framework consists of two axes (X, and Y, or horizontal and vertical). The Y-axis represents the levels of achievements (weights, i.e., low medium and high), while the X-axis represents the complexity and maturity aspects to be rated or weighted. This section introduces the results that we have drawn from the survey using the analysis framework.

We have found six important results from the survey. First, none of the systems reviewed achieve a high level in all aspects. Some systems are better than others in their overall achievement. Second, all systems provide low-level achievement in the open services levels aspect. Third, all systems provide medium level of achievement in distribution, architecture openness, interoperability and maturity aspects. These systems vary in the set of sub-aspects that they satisfy to gain the medium level of achievements. Fourth, all systems provide a high level of achievement in the open data abstraction aspect. Fifth, for other aspects (structure awareness and open structure abstraction) the reviewed systems provide different levels of achievements ranging from low to high, which is an expected result. And finally, in six aspects out of the overall eight aspects in the analysis framework, all the surveyed systems share the same levels of achievements.

From the previous results we can observe that although many of the surveyed OHSs have preformed well in some of aspects, none of them has addressed all aspects.

## 6 A Generic Building Block for CB-OHSs

To help solve this problem we have proposed a building block called BB that addresses all the aspects. The level of achievements of systems built upon this BB will be acceptable in all OHSs aspects. The proposed BB takes into consideration a single component of the OHS environment (services layer component) and then it generalizes this component to be applicable for other components/services as shown in Fig. 1.

Fig. 2 shows the BB for Component Based Open Hypermedia Systems (BB/CB-OHSs). This BB represents only a single component/service of a system/environment, meaning we will have multiple views of the BB according to the number of services we support. The BB consists of the following main components:

- *Service component:* This part is divided into two subparts: each service in BB is comprised of two main parts. The first part os the common part, which includes common components that all services contain, such as the capability to communicate with the naming and location services, the internal message format for the whole BB. The second part is the service unique-core part, which concerns the specific operations of that service (service specific features and tasks).

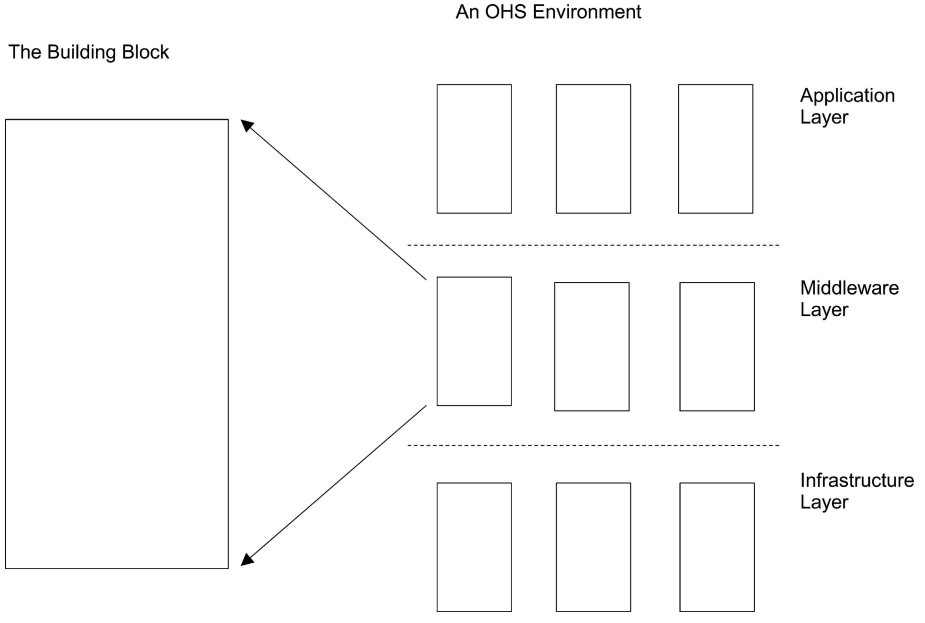



**Fig. 1.** The relationship between the proposed building block and an OHS environment

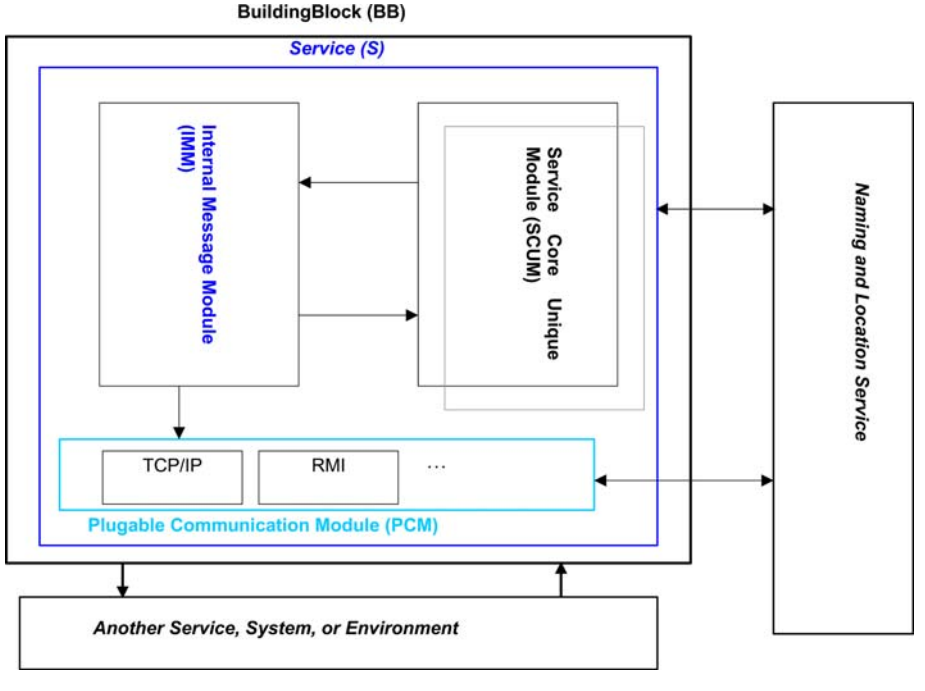



**Fig. 2.** Proposed building block Block for Open Hypermedia Systems (BB/CB-OHSs)

- *Pluggable Communication Module (PCM):* The pluggable communication module is the part (PCM-class), which facilitates the communication between the BB and external BB (such as a service in another BB or system).
- *Internal Message Module (IMM):* This part keeps and manages incoming and out going messages.

### 6.1 BB Advantages

We believe that our BB has several advantages. It improves most of the OHSs aspects especially distribution, interoperability, maturity. This is because of the relations between these aspects. For example, the BB support for component based technology contributes to the improvement of all aspects. The BB also supports an open set of communications protocols, through the provision of the PCM, which is extensible and easier to add to. This contributes to the distribution, architecture openness, interoperability, well definedness and open data abstraction aspects. The BB supports reuse, flexibility, and openness since it uses components that can be extended, add to/update or delete and hence it supports rapid prototyping. We believe that the BB is general and can support different application domains and requirements (since the implementation details is left to the choice of the developer). Finally, the BB Supports different architectures (can be implemented as peer-to-peer or client-server as desired).

### 6.2 BB Improvements on Aspects

The distribution aspect is improved because the BB is component-based; this will help in distribution of all/parts of the BB. Moreover the BB supports an open set of communication protocols; this will facilitate communication between distributed services, applications, and environments/systems. Finally, the implementation language is platform independent. It can run on different operating systems, machines, and network topologies.

The proposed BB improves the interoperability aspect because it is component-based and supports open set of communication protocols (facilitates and achieves all interoperability types).

The BB support of open set of communication protocols, component-based technology, object naming scheme, reuse, use of Java as implementation language (which is popular, object-oriented, platform independent) and provision of services that are general enough to support different type of services and terms will improve the well definedness aspect.

The architecture openness aspect will be improved through the support of component-based technology and extensibility. Hence, easier insertion/deletion of components and the introduction of a new service may only require the extension of one or more of the existing extensible components (open and flexible architecture), reducing implementation time and effort.

Regarding the improvement of the open service levels, the BB can be used as a testbed for experimenting the provision of open services levels. Since services can be added to the system as components, these services can be decomposed into smaller components as desired.

Since the BB supports an open set of communication protocols, this makes easier the communication with other systems, WWW, and applications that support different data formats. Also, the BB support of the reuse facility can help in fast prototyping.

The object-oriented implementation language and support of the object notion will partially improve the achievement in open structure abstraction and structure awareness aspects, since the structure can be implemented and treated as objects.

No service specific model is enforced (architecture or data model). Each BB can have a different model. Additionally, services (SCUMs) within the same BB can have different models, but the problem of understanding each other model will arise. Information about different services models can be placed at the naming service to overcome this problem.

## 7 Reusability

We refer to reusability as the ability to use all/part of an existing code in the development of a new service or component. As the amount the of reused code increases the time and efforts needed to add a new component is decreased and a fast integration and development is achieved [5].

The PCM can be reused either as a whole module or some units of the module can be used as desired. Specification and extensible classes the PCM can be used or modified to implement a new communication protocol. Specifications and extensible classes of SCUM can be used in developing new service/component in addition to the main communication methods (send message, get message, and contact the naming and location service). A whole SCUM can be reused if the service is needed to be replicated in different systems or hosts provided that they all use the same model.

The interface between the SCUMs and the PCM is also reusable. Additionally, the methods that contact the naming and location service to register a new service, to check for a service host and communication protocols when classifying and cashing messages (into internal and external) or registering a service are all reusable.

None of the available systems introduce an implemented PCM that directly supports a set communication protocols (RMI, TCP/IP, SOAP) for communication with other environments/systems. Our model uses dynamic location of services and hosts. Services can change their locations, hosts or communication protocols (ComType) at any time after sending a message, without waiting for the reply. Services can also ask to forward the answer to another service.

The BB can be used to develop services and systems as well as applications. Existing services can still use any newly introduced communication protocol. Restrictions are provided to make sure that the developed service or communication protocol complies with the BB. Incorporated naming and location services support distribution. The BB also supports heterogeneity of programming languages.

The proposed BB is applicable to OHSs, and unique because to date no one has proposed such an environment. Therefore, it can improve achievements of the OHSs developed based on this BB.

## 8 Relations Between Aspects

We think that there are tight relations between the interoperability, architecture openness, distribution, and system well defined aspects achievements. Any improvement/raise of achievement in one of the above aspects will affect directly or indirectly the achievements levels in others, here we provide some examples:

*Example 1.* Improvement of the interoperability aspect achievement between the system components and the system with the integrated applications, WWW, and other systems will increase the system distribution aspect. In order to improve the interoperability, we may use different communication protocols or one of the components framework technologies. This will improve system well defined, architecture openness and distribution as well.

*Example 2.* To improve the well defined, we may use the proposed OHSWG architecture; which is layered architecture. This may improve the architecture openness aspect achievement, as well as the interoperability between system components.

*Example 3.* To improve distribution, naming and location services may be introduced. This may improve both the well defined and interoperability of the system.

Based on the way the proposed BB addresses the individual aspects (direct and indirect) we have divided them into two categories, namely:

*Directly addressed aspects.* This group of aspects represents all parts of the BB except those at the unique core part. The set is directly addressed and improved by the BB. This category includes the following set of aspects: distribution; interoperability; well-definedness; architecture openness; and, open data abstraction.

*Indirectly addressed aspects.* This group of aspects represents the unique core part of each service. This set is addressed and improved indirectly by the BB. In order to provide an argument for this part, the BB can be used as testbed to develop some services to achieve this goal. We think that the achievement or improve in this aspects mainly depends on developers choice during implementation of the BB. This indicates that the BB is general (an advantage of the BB) due to this feature of the BB (generality) the choice is left for the developer to decide according to his needs and preference. This category includes the following set of aspects: open service levels; structure awareness; and, open structure abstraction.

The structure awareness and open structure abstractions are placed here because their address cannot be direct unless the developer of the service is intending to have these aspects in his system. For example, a system based on our BB may have different types of structure types that may or may not be first-class depending on the implementation and the choice of the developer.

# 9 Related Work

## 9.1 Structural Computing and Component-Based Technology

Component-based technology addresses the issue of dividing the system into a set of components. This affects system scalability and facilitates development and interaction between components. While, structural computing [25] deals with the system from a point of a set of structures and relations between these structures. We believe that the proposed BB is a hybrid between these two approaches. It is structure aware, supports an open set of structures at the same time it provides a generic component that can be used to develop all other components needed in a system.

## 9.2 Recent Trends in Design of CB-OHSs

*Component-based approach.* The philosophy of this approach is to support decomposition of the system architecture into components. This increases the system modularity and hence makes it easier to add a new service and modify or delete an existing service at run-time, hence reducing maintenance and development effort and time [20, 21, 22].

*Provision of multiple open services.* This approach divides the services into smaller functionally independent set of services, hence allowing the user to choose the required level. It also improves scalability. Examples of systems that implement this approach are HOSS, Construct and Callimachus system [22].

*Structural computing.* Structural computing is a computation philosophy that separates data and structures; it also allows the definition of behaviors separately [12]. This philosophy defines computations over the general structure element named as behaviors.

*Standardization.* The Open Hypermedia Systems Working Group has made many efforts to support standardization in the community practice. The result of these efforts are: the standard three layer reference architecture, the Open Hypermedia Protocol (OHP), as well as many attempts to propose frameworks for application integration, versioning, collaboration, and integration with the WWW by different OHSWG members. Standardization encourages adoption, reuse and interoperability.

*Provision of development tools to support rapid prototyping.* As a recent trend in OHS development, system designers and developers tend to provide some tools and services to help develop new services and systems (rapid prototyping), hence lowering the entry-barriers, reducing the effort, time, and knowledge (programming skills, distribution and synchronization issues) required to develop a service/system. This approach was first seen in the Hyperform and HOSS environments [12, 20, 21].

*Integration of different hypermedia domains.* A recent trend in structural computing and OHS is the integration of hypermedia domains. The aim of this integration is to allow users to combine and use structural services for different hypermedia domains. Examples for this approach include Fundamental Open Hypermedia Model (FOHM) [17, 18]. Examples of projects or systems implemented this approach include EXTERNAL and the XCHIPS system [26], FOHM [17, 18], and the Auld Linky [27].

## 10 Conclusions and Future Work

In this paper, we have presented the results drawn from our previous survey of some prominent OHSs, including the aspects considered in carrying the survey. We have found that none of the surveyed systems addressed all the aspects. We therefore proposed a BB that is general enough and flexible to help in addressing all aspects. Moreover, the proposed BB can be used as a testbed/design space to experiment/enhance the achievement of the developed system in all aspects.

We believe that the proposed BB can potentially improve the performance of the CB-OHSs in all aspects discussed above, and therefore advance the state of art in this research area. The proposed solutions in this paper dealt with the problem from the conceptual view through the survey, and the BB into application/implementation of the framework and the BB.

Currently the proposed BB is under active development (using Java language). Specifically, we are working on the development of the event queue and the PCM. The entire BB (the first version) is expected to be ready in the near future.

In our future work, we are planning to populate the BB through the development of some services to provide evidence for the improvement that the BB provides for all aspects (especially, the set of indirectly addressed aspects). We think that the open service levels aspect needs more attention of the research community to figure out the possible meaningful levels of services in today OHSs services.

Although we have given a set of aspects in the analysis framework, these aspects cant be seen as unrelated to each other. An improvement in one aspect affects some others aspects as well (as discussed above).

We believe that providing much effort in designing the architecture of OHSs will highly contribute to the elimination of many issues faced by currently available systems. In fact, this conclusion agrees with the standardization efforts of the OHSWG. Our approach is more robust because it provides a whole BB that can be implemented in many different ways according to the choice of the developers.

In terms of Engelbart's A-B-C work level description, we are trying to help systems designers and developers so we are working in B-level. Also the use of the developed services will affect users indirectly (level B, and C).

## References


1. Bush, V.: As we may think. Atlantic Monthly **176** (1945) 101–108
2. Akscyn, R., McCracken, D.: KMS: a distributed hypermedia system for managing knowledge in organizations. Communications of the ACM **31** (1988) 820–835
3. Halasz, F.G.: Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. Communications of the ACM **31** (1988) 836–852
4. Yankelovich, N., Haan, B., Meyrowitz, N., Drucker, S.: Intermedia: the concept and the construction of a seamless information environment. IEEE Computer **21** (1988) 81–96
5. Nürnberg, P.J., Leggett, J.J., Schneider, E.R., Schnase, J.L.: Hypermedia operating systems: a new paradigm for computing. In: Proceedings of the Seventh ACM Conference on Hypertext (HT 96), Washington, DC, USA, ACM Press, New York, NY, USA (1996)
6. Grønbæk, K., Trigg, R.H.: Design issues for a Dexter-based hypermedia system. In: Proceedings of the European Conference on Hypertext 1992 (ECHT '92), Milano, Italy, ACM Press, New York, NY, USA (1992) 191–200
7. Grønbæk, K., Hem, J.A., Madsen, O.L., Sloth, L.: Designing Dexter-based cooperative hypermedia systems. In: Proceedings of the Fifth ACM Conference on Hypertext '93 Conference, Seattle, WA, USA, ACM Press, New York, NY, USA (1993) 25–38
8. Halasz, F.G., Schwartz, M.: The Dexter hypertext reference model. Communications of the ACM **37** (1994) 30–39
9. Grønbæk, K., Trigg, R.H.: Toward a Dexter-based model for open hypermedia: unifying embedded references and link objects. In: Proceedings of the the Seventh ACM Conference on Hypertext, Washington, DC, USA, ACM Press, New York, NY, USA (1996) 149–160
10. Grønbæk, K., Bouvin, N.O., Sloth, L.: Designing Dexter-based hypermedia services for the World Wide Web. In: Proceedings of the Eighth ACM Conference on Hypertext, Southampton, United Kingdom, ACM Press, New York, NY, USA (1997) 146–156
11. Davis, H., Hall, W., Heath, I., Hill, G., Wilkins, R.: Towards an integrated information environment with open hypermedia systems. In: Proceedings of the European Conference on Hypertext 1992, Milano, Italy (1992) 181–190
12. Nürnberg, P.J.: HOSS: an environment to support structural computing. PhD thesis, Texas A&M University, College Station, TX, USA (1997)
13. Wiil, U.K., Leggett, J.J.: The hyperdisco approach to open hypermedia systems. In: Proceedings of the the Seventh ACM Conference on Hypertext, Washington, DC, USA, ACM Press, New York, NY, USA (1996) 140–148
14. Wiil, U.K.: Evaluating hyperdisco as an infrastructure for digital libraries. In: Proceedings of the 1998 ACM Symposium on Applied Computing, Atlanta, Georgia, USA, ACM Press, New York, NY, USA (1998) 491–497
15. Anderson, K.M.: Supporting industrial hyperwebs: lessons in scalability. In: Proceedings of the Twenty-First International Conference on Software Engineering, Los Angeles, CA, USA, IEEE Computer Society Press, Los Alamitos, CA, USA (1999) 573–582
16. Anderson, K.M.: The extensibility mechanisms of the Chimera open hypermedia system. Journal of Network and Computer Applications **24** (2001) 75–86

17. Millard, D.E., Moreau, L., Davis, H.C., Reich, S.: FOHM: a fundamental open hypertext model for investigating interoperability between hypertext domains. In: Proceedings of the Eleventh ACM on Hypertext and Hypermedia, San Anontio, TX, USA, ACM Press, New York, NY, USA (2000) 93–102
18. Ridgway, N., DeRoure, D.: RTSP+FOHM: applying open hypermedia and temporal linking to audio streams. In: Revised Papers from the Third International Workshops OHS-7, SC-3, and AH-3 on Hypermedia: Openness, Structural Awareness, and Adaptivity, University of Aarhus, Århus, Denmark, Springer-Verlag, Heidelberg, Germany (2001) 71–81
19. Tzagarakis, M., Vaitis, M., Papadopoulous, A., Christodoulakis, D.: The Callimachus approach to distributed hypermedia. In: Proceedings of the Tenth ACM Conference on Hypertext and Hypermedia: Returning to our Diverse Roots, Darmstadt, Germany, ACM Press, New York, NY, USA (1991) 47–48
20. Wiil, U.K., Hicks, D.L.: Tools and services for knowledge discovery, management and structuring in digital libraries. In: Proceedings of the Eighth ISPE International Conference on Concurrent Engineering: Research and Applications (CE 2001), Anaheim, CA, USA (2001) 580–589
21. Wiil, U.K.: Development tools in component-based structural computing environments. In: Proceedings of the Seventh Workshop on Open Hypermedia Systems. Volume 2266 of Lecture Notes in Computer Science., Århus, Denmark, Springer-Verlag, Heidelberg, Germany (2001) 82–93
22. Wiil, U.K., Hicks, D.L., Nürnberg, P.J.: Multiple open services: a new approach to service provision in open hypermedia systems. In: Proceedings of the Twelfth ACM Conference on Hypertext (Hypertext 2001), Århus, Denmark, ACM Press, New York, NY, USA (2001) 83–92
23. Anderson, K.M., Sherba, S.A., van Lepthien, W.: Structure and behavior awareness in Themis. In: Proceedings of the Fourteenth ACM Conference on Hypertext and Hypermedia, Nottingham, United Kingdom, ACM Press, New York, NY, USA (2003) 138–147
24. Anderson, K.M., Sherba, S.A., van Lepthien, W.: Structural templates and transformations: the Themis structural computing environment. Journal of Network and Computer Applications **26** (2003) 47–71
25. Wiil, U.K., Hicks, D.L.: Providing structural computing services on the World Wide Web. In: Revised Papers from the Third International Workshops OHS-7, SC-3, and AH-3 on Hypermedia: Openness, Structural Awareness, and Adaptivity, University of Aarhus, Århus, Denmark, Springer-Verlag, Heidelberg, Germany (2001) 160–171
26. Wang, W.: Visualizing and interacting with hypermedia-based process-centric enterprise models. Journal of Network and Computer Applications **26** (2003) 73–93
27. Millard, D.E.: Discussions at the data border: from generalized hypertext to structural computing. Journal of Network and Computer Applications **26** (2003) 95–114

# Applying Information Visualisation Techniques to Spatial Hypertext Tools

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.** Organising information is an important knowledge work activity that is frequently used in the work place and at home. Even though this task is an every day activity, it is nontrivial. Some tools exist that take advantage of our spatial and visual intelligence, but have some difficulties with creating a satisfying visualisation for the information. Visualisations built by users may not show the information in the most useful way, so important facts may not emerge in time, or at all. Information visualisation suggests possible methods by which to visualise various types of information. However, it focuses on existing and explicit structures. This paper suggests combining spatial hypertext with information visualisation techniques to allow users to organise their information more effectively.

## 1 Introduction

Structuring information is an important task used in many analysis situations. It is a difficult and time-consuming task that can change over time. The process happens in several stages. Firstly, information is collected. Secondly, it is organised in some way (e.g., alphabetically, chronologically, by topic etc.). This process then continues until a satisfactory result is achieved. In some cases, it can be reported or presented afterwards [1].

The growth of the Internet as a public information resource means that it is easier to access and gather information quickly. An advantage of this is the ability to keep a digital copy of that information for future reference. This means that users have potentially large personal libraries of information on their computers. This amount of information is difficult to organise within a hierarchical file structure (e.g., Windows file system). Some tools exist that allow users to organise their information informally [2]. However, difficulties can occur when trying to navigate that information.

Information visualisation tools exist that allow users to navigate around large information spaces. However, most visualisation tools are designed to work with existing and explicit structures. Existing tools focus mainly on navigation of information, not on interaction with it.

The focus of our work is, instead, on implicit and emergent structures and interaction of information. By looking at the techniques already available in

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 85–93, 2004.

traditional information visualisation tools, it should be possible to transfer some of those techniques into spatial hypertext tools. Both have a similar problem i.e. visualising large amounts of disparate information is difficult, so a similar solution should exist for both. Different categories of information exist [3] and can be paired with various visualisation techniques to produce the most effective result. If the information to be organised manually fits one of those defined categories, techniques from that visualisation method could be used for spatial hypertext tools.

The remainder of this paper is organised as follows: Sect. 2 describes and compares spatial hypertext and information visualisation tools. Section 3 describes the tasks that information visualisation tools are expected to do, as well as the types of information available and appropriate visualisation methods. It also outlines a possible tool. Section 4 discusses some existing spatial hypertext and information visualisation tools. Section 5 discusses potential furture work. Section 6 offers some conclusions.

## 2 Spatial Hypertext Versus Information Visualisation

Some similarities exist between spatial hypertext and information visualisation. Information to be organised is similar. One difference is in if a user of computer (or both) organise it.

**Spatial Hypertext.** Hypertext research looks at how to present and represent material electronically that is too complex to represent on paper [4]. Structures discussed in the hypertext domain include: associative (navigational); taxonomic; and, spatial. Associative structure links individual documents together. Taxonomic structure categorises information and represents them as a hierarchy. Spatial structure allows users to organise with visual attributes, such as proximity, colour, shape and size.

When considering how users interact with paper, it can be seen that users take advantage of the space that surrounds them. Observations in offices [5] describe how users structure their paper. The ease/difficulty of this is dependent on several factors, including the amount to be organised and type of information to be organised. The use of an unstructured "pile", loosely gathered collection of paper, is important to users. Other studies found that users were unable to remember file names precisely, but were given clues from the objects that surrounded them (e.g., a filing cabinet holds archives of related information [6].) This kind of spatial organisation was also observed when using Windows and Macintosh filing systems [6].

Spatial hypertext tools work effectively with information that is organised implicitly. As no rules are defined by users beforehand, users organise to suit their own needs.

A limitation of spatial structures is as the amount of information increases, it becomes more difficult to see both detail and an overview at the same time.

Possible solutions to this include, introducing multimodal cues [7], increasing dimensions [8] and incorporating information visualisation techniques into spatial hypertext tool. This paper focuses on the latter option.

**Information Visualisation.** When the amount of information becomes large enough in users computers, it becomes increasingly difficult to navigate around information, to see the details, as well as an overview. Information visualisation techniques takes advantage of users' perceptual characteristics to adapt information into a graphical representation. In these tools, as screen size is limited, each pixel must be taken advantage of. Various tools and techniques exist allowing users to look at information, in order to see patterns emerge.

Visualisation techniques can be used for different occasions [9]. This is summarised in Table 1.

**Comparison.** Information visualisation tools concentrate on automatic organisation. They are usually completed mostly by a computer. Rules are set by users explaining how the information is to be organised beforehand. Difficulties occur when objects belong in more than one place. However, as it is relatively fast to change the organisation, these tend to be short-lived and created "on-the-fly".

Spatial hypertext tools concentrate on manual organisation. This is usually undertaken by users. Knowledge of a given topic develops over time and a better understanding of a given problem develops over time. Users are often unable to fully explain their choices, or why a document is more related to one folder than another. This kind of organisation tends to be long-lived; that is, it has taken time to organise, so there is more reluctance to throw it away. More emphasis is placed on remembering connections instead of defining connections mechanically.

Both information visualisation and spatial hypertext tools takes advantage of users' perpetual abilities to scan, recognise, and recall images rapidly with

**Table 1.** Levels at which visualisation can be used

| | **Contents** | **Primary Use** |
|---|---|---|
| Infosphere | Information outside the users' environment. | Place to find information needed for work. |
| Information workspace | Information with which the user is interacting as part of some activity. | Place to hold work in progress. Used for reducing cost of work, reminding user of work materials. |
| Visual knowledge tools | A data set. | Substrate into which data is poured and/or tool for manipulating it. Used for pattern detection, knowledge crystallisation. |
| Visual objects | One of more data sets packaged for convenience. | Packaging of data (data often known in advance). Used to enhance objects of interaction. |

both moving work from the cognitive to the visual perceptual systems. Both tools may have similar information to organise, with similar outcomes desired. Therefore, it should be possible to transfer some techniques from information visualisation to spatial hypertext.

## 3 Adapting Spatial Hypertext Tools

Before creating a mixed spatial hypertext/information visualisation tool, it is necessary to understand what tasks this tool should be able to perform and what kind of visualisation technique to use. Section 3.1 describes the tasks that all visualisation tools are expected to perform. Section 3.2 describes the types of information that are available. Section 3.3 describes various types of visualisation techniques. Section 3.4 describes a possible tool that combines spatial hypertext elements with the relevant parts of information visualisation. This includes what visualisation to use.

### 3.1 Tasks Supported

An important part of most information visualisation tools is the principle "Overview first, zoom and filter, details on demand" [3]. When designing a tool that supports information visualisation, the following tasks should be supported [3]:

- **Overview.** Gain an overview of the entire collection. Strategies include: fisheye; context plus focus; content and view; etc.
- **Zoom.** Zoom in on items of interest. Users typically are interested in a part of a collection. Strategies include: smooth zooming ...
- **Filter.** Filter out uninteresting items. Dynamic queries applied to the items in the collection. Strategies include: use of sliders; buttons and other control widgets combined with rapid display update.
- **Details on demand.** Select an item or group of items and obtain details when needed. The usual approach is to simply click on an item to get a pop-up window with values of each of the attributes.
- **Relate.** View relationships between items.
- **History.** Keep a history of actions to support undo, redo and progressive refinement.
- **Extract.** Search for particular subsets of items. Ability to save searched for information to a file.

### 3.2 Types of Information

Information can be divided into several categories depending on its attributes. The category it belongs to indicates possible visualisation techniques that best suit it [3]. Some of the information types include:

- **1-dimensional.** These are linear data types (e.g., alphabetical lists of names organised sequentially.)
- **2-dimensional.** This is also known as spatial (e.g., planar or map data including floorplans, newspaper layouts, etc.)
- **3-dimensional.** real world objects such as molecules, the human body, and buildings have items with volume and some potentially complex relationship with other items. e.g. 3D computer graphics and computer-assisted design are large topics, but information visualisation efforts in three dimensions are still novel.
- **Temporal.** The difference between 1D and temporal is that temporal data have a start and finish time and that items may overlap.
- **Multidimensional.** Metadata attributes such as type, size author, modification date, etc. Items with $n$ attributes become points in *n-dimensional* space. Most relational and statistical databases are conveniently manipulated as multi-dimensional data in which items with n attributes become points in a n dimensional space. The interface may be 2D scattergrams with additional dimensions controlled by a slider.
- **Tree.** Any kind of tree structure (e.g., file systems on computers, library classification schemes, etc.)
- **Networks.** Graphs (e.g., hypermedia node-link graphs, webs, etc.)

### 3.3 Visualisation Techniques

Each type of information responds better to different types of visualisations. Some are more appropriate than others.

- **1-dimensional.** Visualisation methods include using bifocal displays that provide detailed information in the focus area and less information in the surrounding context area.
- **2-dimensional.** Methods include multiple-layer approach, with each layer being 2D.
- **3-dimensional.** Techniques such as overviews, landmarks, perspective, stereo display, transparency and colour coding.
- **Temporal.** Similar to 1D, includes perspective wall and lifelines
- **Multidimensional.** Represented with scattergrams with each additional dimension controlled by a slider.
- **Tree.** Several techniques exist as this is a frequent storage system (e.g., file folder system in desktop.)
- **Networks** some tree visualisations, graph displays.

### 3.4 Possible Tool

***Tasks.*** At present some spatial hypertext tools support overview, zoom, relate, redo and organise. However filter and extract are not implemented in some tools.

***Types of information.*** Information being organised in a spatial hypertext tool may fall into any of the described categories. The kind of the information to organise is dependent on the user, and is always unknown to the tool. Spatial hypertext tools should support all knowledge work users.

However, in this case the organisation (structure) of the workspace is in some ways more important than the data itself. A map, which is an example of a 2D information type, can be represented as a node that may or may not be connected to other maps. For example, a website may contain the maps of each country in Europe. Each map shows its information in a 2D way. Each map may have a relation to other maps, for example, what are the neighbouring countries? This kind of information is an example of a network. Therefore, even though the information may fall into any category, when it is introduced into the spatial hypertext application, it is changed to being of type network. Therefore, only network visualisation techniques will be discussed.

***Visualisation techniques.*** Networks may be visualised in various ways. Some of those techniques are as follow;

- **Using hierarchical visualisation techniques.** One method is to reduce the network(graph) into a tree and then use one of the many hierarchical visualisation methods, e.g.
  - Classic tree drawings.
  - Tree browsers.
  - Tree maps.
  - Radial approaches.
  - Cone trees.
  - Landscapes.
  - Hyperbolic browsers.
  - etc.
- **Layered methods.** Layered methods are used for drawing directed graphs, which are graphs having general flow or direction.
- **Force-directed placement.** These methods draw on physical analogies and are applicable to general graphs, without any prior knowledge of structural properties.
- **Energy-based placement.** A net force indicates which direction to move an object in order to reduce the forces acting upon it. This is equivalent to minimising an implicit internal energy model of the system. These techniques attempt to minimise this residual energy directly.
- **Semantic zooming.** Semantic networks are networks of associations between concepts. Semantic networks are often too large to visualise all at once in a single visualisation. Semantic zooming techniques attempt to manage this complexity by visualisation only a small part of the network at any one time, but providing for fluid motion between related concepts.

## 4 Related Work

The following tools are used with automatically created information organisations, such as the Internet or already arranged hierarchical file systems. They are examples of how to visualise networks.

### 4.1 Caliph and Emir

Caliph & Emir are MPEG-7 base prototypes for digital photo and image annotation and retrieval supporting graph like annotation and content based image retrieval [10].

Caliph supports the creation of new metadata as well as reading the existing information that already exists in digital photographs. Semantic information about the image is presented as a directed graph, where the nodes reflect semantic objects, locations, agents, states, times or concepts and the edges define the relations between these semantic entities. To enhance retrieval efficiency content-based metadata is extracted and new instances of the image are created for faster visualisation, like thumbnails. The MPEG-7 description consists of the following parts: metadata description, creation information, media information, textual annotation, semantics, visual descriptors.

Emir is an experimental metadata based image retrieval tool that supports retrieval in file system based photo repositories created with Caliph. Different types of retrieval mechanisms are supported, such as content based image retrieval, and searching through XPath statements, etc.

### 4.2 Infosky

InfoSky is a system enabling users to explore large, hierarchically structured document collections [11]. Similar to a real-world telescope, InfoSky uses a planar graphical representation with variable magnification. Documents of similar content are placed close to each other and are visualised as stars, forming clusters with distinct shapes. Textual labels are displayed dynamically during navigation, adjusting to the visualisation content. Navigation is animated and provides a seamless zooming transition between summary and detail view. Users can map metadata such as document size or age to attributes of the visualisation such as colour and luminance. Queries can be made and matching documents or collections are highlighted.

### 4.3 IsaViz

IsaViz is a visual environment for browsing and authoring Resource Description Framework(RDF) models represented as directed graphs [12]. It features a 2.5D user interface allowing smooth zooming and navigation in the graph and the creation and editing of graphs by drawing ellipses, boxes and arcs.

## 5 Future Work

At present we are working on a bi-modal (audio and visual) spatial hypertext tool. We aim to take advantage of other abilities that users have (e.g., audio memory) in order to reduce visual overload. The project will increase and develop the number of audio cues.

This paper focuses on combining relevant elements from information visualisation and spatial hypertext research. It identifies the similarities between the two areas, and what may be transferred. Once it is known what elements can be used, the next task is, using multimedia principles, to combine the information visualisation/spatial hypertext principles with multimodal techniques in order to create an effective multimodal spatial hypertext tool. A prototype will then be designed, implemented and evaluated.

## 6 Conclusions

Spatial hypertext tools take advantage of users' spatial and visual intelligence. Users' organise their information informally, without expressing rules beforehand. This has several advantages over more rigid hypertext structures. Some advantages include aiding in the explanation of ideas to colleagues and supporting structured ambiguity. Once an information workspace is created, users then navigate around their areas trying to find patterns and gain new insights. Spatial hypertext tools are not so well suited to this task. One solution is to introduce relevant information visualisation techniques.

Similarities between spatial hypertext tools and information visualisation tools exist (e.g., the information to be organised is the same), the difference is whether it is organised by a person or a machine. In spatial hypertext tools, users' organise without defining explicit rules. In information visualisation tools, computers organise with pre-defined rules. At present, the main focus for information visualisation tools is on creating an automatic visualisation from predefined rules.

This paper considered how information visualisation techniques could apply to spatial hypertext tools. It discussed various usability aspects of both tools, and described a possible tool that combines both. As so many similarities exist, it is possible to transfer some techniques, and therefore improve spatial hypertext tools.

## References

1. Russell, D.M., Stefik, M.J., Pirolli, P., Card, S.K.: The cost structure of sensemaking. In: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press (1993) 269–276
2. Shipman, F.M., Hsieh, H., Maloor, P., Moore, J.M.: The visual knowledge builder: a second generation spatial hypertext. In: Proceedings of the twelfth ACM conference on Hypertext and Hypermedia, ACM Press (2001) 113–122

3. Shneiderman, B.: The eyes have it: A task by data type taxonomy for information visualizations. In: Proceedings of the 1996 IEEE Symposium on Visual Languages, IEEE Computer Society (1996) 336
4. Nelson, T.H.: Complex information processing: a file structure for the complex, the changing and the indeterminate. In: Proceedings of the 1965 Twentieth National Conference, ACM Press (1965) 84–100
5. Malone, T.W.: How do people organize their desks?: Implications for the design of office information systems. ACM Trans. Inf. Syst. **1** (1983) 99–112
6. Barreau, D., Nardi, B.A.: Finding and reminding: file organization from the desktop. SIGCHI Bull. **27** (1995) 39–43
7. McGookin, D.K., Brewster, S.A.: Fishears: The design of a multimodal focus and context system. In: Proceedings of the IHM HCI. (2001)
8. Robertson, G., Czerwinski, M., Larson, K., Robbins, D.C., Thiel, D., van Dantzich, M.: Data mountain: using spatial memory for document management. In: Proceedings of the Eleventh Annual ACM Symposium on User interface software and technology, ACM Press (1998) 153–162
9. Card, S., Robertson, G., York, W.: The web-book and the web forager: An information workspace for the world-wide web. In: Proceedings of the CHI'96, ACM Conference on Human Factors in Computing Systems, ACM Press (1996) 111–117
10. http://sourceforge.net/projects/caliph emir/: Caliph & emir (2004)
11. Andrews, K., Kienreich, W., Sabol, V., Becker, J., Droschl, G., Kappe, F., Granitzer, M., Auer, P., Tochtermann, K.: The infosky visual explorer: exploiting hierarchical structure and document similarities. Information Visualization **1** (2002) 166–181
12. http://www.w3.org/2001/11/IsaViz/: Isaviz (2001)

# An Agenda for Structural Computing Research

Uffe Kock Wiil[1], David L. Hicks[2], and Peter J. Nürnberg[2]

[1] Mærsk Mc-Kinney Møller Institute, University of Southern Denmark,
Campus 55, 5230 Odense M, Denmark
ukwiil@mip.sdu.dk
[2] Department of Software and Media Technology, Aalborg University Esbjerg,
Niels Bohrs Vej 8, 6700 Esbjerg, Denmark
{hicks, pnuern}@cs.aue.auc.dk

**Abstract.** Structure plays an important role in knowledge work. The overall goal of structural computing research is to provide effective support for knowledge workers. Structural computing research has reached a level of maturity, where it is relevant and essential to define and discuss common research directions. This paper presents an agenda for structural computing research. A conceptual architecture for a structural computing environment is presented to serve as a point of reference (framework) for discussing important issues facing the structural computing research community.

## 1 Introduction

Structural computing is a relatively new research direction that grew out of the very successful work on open hypermedia systems (OHS). OHS research was a focal point in several major hypermedia research groups in the 1990's [4, 8, 9, 43]. A successful workshop series ran from 1994 onwards [44–48]. In 1996, the Open Hypermedia Systems Working Group (OHSWG) started their work towards common de-facto standards for system architectures, service definitions, and application integration (interoperability) [6, 30].

The term "structural computing" was coined in 1997 [26]. The structural computing idea was driven by the philosophy of the "primacy of structure over data" [26]. In a structural computing environment, structure is given first class status allowing it to be manipulated independently of data. There are two important defining factors of structural computing environments. Firstly, *structure awareness* is pushed deep into the environment – in one case as deep as the underlying operating system [27]. Secondly, a structural computing environment provides support for *multiple structure domains* within the same environment such as support for associative linking (navigational domain), spatial ordering (spatial domain), and classification (taxonomic domain).

Structure plays an important role in knowledge work [24]. The overall goal of structural computing research is to provide effective support for knowledge workers. Structural computing is an approach to designing and implementing computing environments where structure plays a central role at all levels in the environment (structure awareness) and where different types of structure used in different phases

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 94 – 107, 2004.

or types of knowledge work can co-exist and interoperate (multiple structure domains).

Knowledge workers, who make use of structural computing environments, may use different knowledge tools that each supports a specific subtask in their knowledge work. For instance, if a knowledge worker is given the task of reviewing (understanding, organizing, and presenting) a specific body of knowledge, she may use a spatial ordering tool in the first phases of her work where no formal understanding of the knowledge entities exists. In the next phases she may add metadata to customize the knowledge entities. She may also add links to associate individual knowledge entities. Finally, she may use a taxonomic tool to classify the knowledge entities to reflect her obtained understanding of the knowledge entities.[1]

In 1999, a workshop series on structural computing was initiated [23, 28–29]. By 2002, this series had been replaced by the Metainformatics Symposium [11, 22]. The first structural computing environments are now well on their way (e.g., FOHM [16–17], Callimachus [32, 35], Themis [2–3], XCHIPS/EXTERNAL [36–37], IUHM [14, 18], and Construct [39, 41]). The work in the research community has reached a level of maturity, where it is relevant and essential to define and discuss common research directions for the future.

This paper presents an agenda for structural computing research. Section 2 presents a conceptual architecture for structural computing environments and presents existing prominent structural computing work. Important issues facing the structural computing research community are discussed in Section 3. Section 4 concludes the paper.

## 2 Structural Computing Environments

This section presents a conceptual architecture for structural computing environments that will serve as a point of reference (framework) for presenting existing structural computing work (this section) and for discussing the prominent issues faced by the structural computing community (next section).

### 2.1 Conceptual Architecture

As mentioned, structural computing research grew out of OHS research. Many ideas relating to architectures, services, etc. found in structural computing environments can be traced back to OHS research. A prominent example of an "inherited" idea is the layered architectural model with provision of services wrapped in components with well-defined interfaces – inherited from the research on component-based OHS [42]. Therefore, the conceptual architecture presented in this section (Fig. 1) is not much different from the reference architecture defined by the OHSWG [30].

Structure is the fundamental building block in a structural computing environment. The basic structural abstraction (*the structural atom*) is provided by a structure store at the foundation layer. All other structural abstractions in the environment are

[1] A more detailed scenario of the use of structuring tools in knowledge work is presented in [24].

modeled upon the structural atom. Thus, a structural computing environment is structure aware at its most basic layer (the first defining factor). The foundation layer also provides other basic infrastructure services that handle naming and location, access control, event notifications, versioning, and locking (concurrency control).

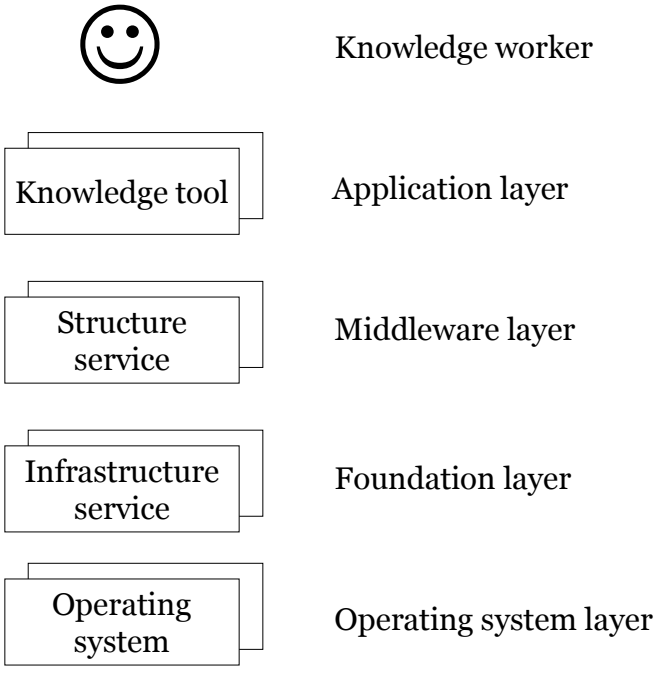



**Fig. 1.** The conceptual architecture for a structural computing environment

The middleware layer hosts multiple structure services that each supports a specific structural domain (i.e., navigational structures for associative linking of knowledge, taxonomic structures for classifying knowledge, or spatial structures to organize knowledge using a spatial metaphor). Thus, multiple structural domains are supported within the same environment (the second defining factor).

Knowledge tools in a structural computing environment use structure service functionality to augment and enhance the service they provide to end users (knowledge workers). For example, an editor application could utilize the facilities of a navigational structure service to provide a basic linking capability for text objects. Similarly, a graphical drawing tool might incorporate the services of a spatial structure service in order to enhance its information management capabilities.

### 2.2 Existing Work

Though a relatively new field, research results have already begun to emerge for the structural computing area [11, 22–23, 28–29, 38]. Research has started on a variety of fronts including the design and development of structural computing environments

and the use of structural computing techniques and technology to support specific application areas. The intention in this section is to examine the overall approach of several structural computing projects to provide a representative sample of the research results that have been reported. Specific details of these projects will be discussed together with the research issues in Section 3.

Researchers at Texas A&M University conducted the HOSS project. It proposed a view of hypermedia that considered it to be a new computing paradigm [27], thereby laying the groundwork for much of what was to become structural computing. The HOSS project sought to integrate hypermedia (structure) into the lowest, most fundamental levels of a computing environment – the operating system.

Researchers at the University of Patras have proposed Callimachus a general-purpose structural computing environment that supports a wide range of structural abstractions [32, 35]. The objective is to support not only a variety of structural domains such as the navigational, spatial, and taxonomic domains, but also provide support for specialization within each domain (sub-domains) and the combination of domains (applications).

Construct is a general-purpose structural computing environment from Aalborg University Esbjerg [41]. A primary goal of the project is to provide a comprehensive environment for supporting structure (e.g., navigational, taxonomic, spatial, metadata, and cooperation). Tools are provided to assist in the development of structure services – UML Tool, which allows structure services to be specified in the UML language, and CSC (Construct Service Compiler), which can generate code based on structure service specifications [40].

The Fundamental Open Hypermedia Model (FOHM) was developed at the University of Southampton [16–17]. FOHM is based on, and is an extension of the OHP (open hypermedia protocol) [6], a standard that was developed by the OHSWG to support the interoperability of hypermedia systems. OHP was specifically targeted to support interoperability within the navigational hypermedia domain. FOHM broadens its applicability to also include the spatial, and taxonomic hypermedia domains. FOHM promotes interoperability between domains.

At the University of Colorado researchers have in the Themis project focused on the application of structural computing technologies and techniques to a particular application area, software engineering [2–3]. They have analyzed a number of software engineering sub-domains in order to generate a general set of requirements that structural computing environments must meet in order to accommodate the software engineering area. Themis has evolved into a general-purpose structural computing environment.

Research at the Fraunhofer IPSI Institute (formerly GMD-IPSI) has been conducted that involves user interface issues of structural computing [36–37]. As part of the project, a graphical user interface has been designed that integrates features from the navigational, spatial, taxonomic, workflow, and cooperation domains. This work is performed under the XCHIPS/EXTERNAL project, which promotes and explores interoperability at the application layer in structural computing environments.

Researchers at the University of Montpellier have worked on the IUHM project [14, 18]. The IUHM (Information Unit Hypermedia Model) is an architectural model

aimed at supporting modeling and implementation of dynamic systems. The focus in the IUHM project is on information integration. Structural computing techniques are used in the management of large-scale extensibility and tailorability. IUHM has been used in the implementation of OPALES – an open, collaborative digital library system for audio and visual data [19].

## 3 Research Issues

This section presents some of the important research issues facing the structural computing community. The following discussion focuses on issues that specifically relate to structural computing research. Several general issues relating to design and implementation of component-based OHS discussed by the OHSWG (such as reference architectures, application integration, and interoperability) [6, 30] and general issues discussed by Nürnberg et al. in their 1998 research agenda for OHS [25] are also relevant for current structural computing research. In a few cases there will quite naturally be overlap between the current structural computing issues and the issues discussed by the OHSWG and the 1998 OHS research agenda. In those cases, the intention in this paper is to provide the latest and most up-to-date view on the issues.

### 3.1 Development Methodology

The structural computing community claims that structural computing environments are different from traditional software environments. Structural computing provides a different perspective on how to understand and develop software environments that includes a new structural dimension – in addition to the well understood data (model) and behavior (functionality) dimensions of traditional software environments.

Developers of software environments have the option to use a formalized methodology that explain in detail a number of steps in the process of software development (i.e., system definition, requirements definition, analysis, design, implementation, test, etc.). Such development methods are mature and are used extensively in the development of various types of software environments. Examples of modern, mature development methods include Object-oriented Analysis and Design [15] and Unified Process [12].

Yet, it is not explained in the structural computing literature how we should approach the difficult tasks of developing structural computing environments. There are currently a number of open issues. What steps are involved in developing a structural computing environment? To what extent can we reuse ideas and tasks from existing modern software development methods? What (if any) tasks in software development have to be handled in a different manner when developing a structural computing environment?

Despite the fact that no structural computing research group has been explicit about their development methodology, structural computing environments are being developed in many different places. Therefore, much (implicit) knowledge about development exists in these research groups. Given the number of structural computing environments available today, it would be wise to start formalizing the

development process. Both existing and new research groups in structural computing would benefit greatly from having a common view on how to develop their environments.

Once a common consensus has been reached on a development methodology, it is possible to think in terms of building development tools that can support different stages in the development process.

Construct is an example of a structural computing project that provides a development environment consisting of different tools that ease the development of new services wrapped in components [40]. New service components are developed in two steps. The first step is to specify the interface (API) of the component in either UML or IDL. These high-level specifications of components are then given as input to the Construct Service Compiler, which generates service skeletons wrapped in components. The second step is to fill-in the semantic parts (methods bodies) of the individual operations in the component interface.

Yet, there is no overall notion of a development methodology in Construct. This "backwards" approach of having development tools but no overall methodology is problematic. The basic problem with the Construct approach is that the development support starts relatively late in the development process, when the developer knows what structures to implement and how to group them into services (components). It is not been made explicit how to take the step(s) from having knowledge about a specific work domain to having a description of what structures should be used to support the work in this domain. The community needs to define how to perform a "structural analysis" of a work domain to identify necessary structural support. Can we reuse ideas from existing software development methodologies that specify how to do domain analysis or do we have to start completely from scratch and create a new development methodology for structural computing?

In summary, the notion of a development methodology for structural computing that specifies and supports the individual steps in the development process, that guides the overall process, and that specifies how to generate standardized documentation as the result of the individual steps is still unchartered territory in structural computing research.

### 3.2 Infrastructure

Computing infrastructure is part of any computing environment. Each type of computing environment has specific needs with respect to infrastructure support depending on the overall task of the computing environment.

The infrastructure support needed in structural computing environments is influenced by at least three things: the overall work tasks (knowledge work), the focus on both structure and data abstractions, as well as the overall evolution in system architectures and technologies.

Knowledge work often involves more than one person. The collaboration in a group of knowledge workers should not be limited in time (same time / different times) and space (same place / different places) [7]. This alone immediately poses several requirements on the infrastructure. The distributed nature of knowledge work

implies the need for basic distribution support such as naming and location services. The collaborative nature of knowledge work implies the need for basic collaboration services such as access control, event notification, and locking. The evolutionary nature of knowledge work implies the need for services that support tracking and maintaining changes over time (versioning).

These infrastructure requirements are not new to the hypermedia field, and it is not surprising to find them in the context of structural computing environments also. One of the defining characteristics of structural computing complicate the provision of these services in structural computing environments compared to general computing environments, namely the high degree of structural awareness. Services that provide access control, locking, event notifications, and versioning need to be able to handle the structural dimension also. Access control should also apply to structural abstractions. A knowledge worker needs to have appropriate access permissions to for instance follow a link and open a knowledge space. Locking should also apply to structural abstractions such as metadata. Versioning of structure becomes critical in order to track the evolution of the knowledge work in a complete manner. Event notifications should also be generated when structural abstractions are created, modified, and deleted. These are just a few examples of the increased complexity of infrastructure services due to the structural dimension in structural computing environments.

The evolutionary trend in structural computing environments that can be traced back to component-based OHS also complicate the provision of infrastructure services. The benefits of the layered architectural model with provision of services wrapped in components with well-defined interfaces (inherited from previous work on component-based OHS) are increased flexibility and increased usability. The drawbacks are less control and increased complexity. Infrastructure services are easier to build for a monolithic system that has full control over data and structure storage, functionality, and presentation. In component-based architectures, data and structure storage, functionality, and presentation are distributed over several autonomous components resulting in less control and increased complexity (e.g., increased communication between components).

Another issue raised by the current architectural approach is that of combination services. This is best illustrated with an example. Consider a structure service that needs to combine two existing infrastructure services (e.g., structure storage and version control) to provide a versioned structural atom. How should this be done? Should the responsibility be placed on the structure service by having this service use the functionality of the structure store and the version service in a certain pattern? Should a new infrastructure service be defined that combined the needed support from the two infrastructure services? Should the two infrastructure services be modified to communicate with each other to provide the needed functionality?

Some of these issues have been addressed by current structural computing projects. Callimachus has experimented with naming and location services [33]. Construct has experimented with collaboration services (sessions, awareness, and events) [39]. However, no structural computing environment has addressed the full range of infrastructure issues yet.

In summary, much work still needs to be done on defining and experimenting with infrastructure services in structural computing environments. It should be possible to reuse many ideas, concepts, and mechanisms from earlier generations of hypermedia systems. Until more experimental work has been made, no one knows the full implications of the increased complexity.

### 3.3 Interoperability

In an environment with multiple structure services (one of the defining factors of structural computing) interoperability plays an important role. The fact that services are decoupled and modularized at all architectural layers raises a number of issues regarding the interoperability of the decomposed services.[2]

From the perspective of the knowledge workers there should be smooth transitions between the different phases in knowledge work. Each phase might include work with different structure services as the knowledge work scenario in the Introduction indicates. This means that the structure manipulated by one structure service should be made available to other structure services. How this should be done is not obvious. One could imagine an approach where structure is *translated* from one type to another. One could also imagine that the structure of one structure service could be *interpreted* by another structure service.

In general, interoperability can occur at all levels in a structural computing system. The XCHIPS/EXTERNAL structural computing environment provides an example of interoperability at the application layer [36–37]. The user interface in XCHIPS/EXTERNAL combines structures from several structure domains in the same interface.

Nürnberg et al. first discussed the issues of interoperability among structure services (middleware layer), specifically the issues involved in having one structure service interpret the structure of another structure service (e.g., a navigational structure service that can interpret spatial structures) [25]. Later, Millard et al. introduced the FOHM model, which addresses interoperability between three particular structure services (spatial, navigational, and taxonomic) by defining a common data model for these three structural domains [17].

Considering interoperability between different infrastructure services (foundation layer) is also relevant. This implies the need for each infrastructure service to be able to interpret (translate) the abstractions manipulated by other infrastructure services. Is this a realistic or even useful goal? For example, consider an access control service that can interpret the abstractions of a locking service. This particular example may make sense, since the fact that a document is locked could be interpreted as a special level of access control denying other people the ability to open the document. Clearly, we cannot yet say that translating from any given infrastructure service to any other is a useful undertaking, but it does seem that at least some of these translations may be useful.

One way to discuss interoperability is to distinguish between horizontal and vertical interoperability [41]. Horizontal (intra-layer) interoperability covers the cases

[2] Parts of this section are based on similar discussions in earlier papers – in particular [25] and [41].

where a service at one layer can interpret (translate) abstractions provided by other services at the same layer. Vertical (inter-layer) interoperability covers the cases where a service at one layer can interpret and use abstractions provided by services at other layers. Vertical interoperability occurs in layered systems that provide different abstractions at different layers.

Atzenbeck et al. has recently started working on the definition of a structure domain interoperability space. The first results are reported in [5].

In summary, interoperability is a complex, mostly unexplored issue. The different interoperability options should be mapped out in detail (e.g., based on the horizontal versus vertical distinction) and various experiments should be made to explore the potential benefits that interoperability between services can provide.

### 3.4 The Structural Atom

A defining characteristic of structural computing environments is the awareness of structure deep in the environment. This characteristic has most often been linked with the existence of a basic structural building block (often called the structural atom) of a structural computing environment. The structural atom is generally thought to be the foundation for all abstractions in a structural computing environment. Much work has been done to address this issue – since every full-blown structural computing environment has a structural atom. Although details of implementation vary among the different approaches, all such structural atoms basically have the ability to both represent structure and to be structured (i.e., participate in other structures).

When designing a structural computing environment it is important to decide where (how deep) in the system the awareness of structure should be placed. The answer probably depends on the task(s) that the structural computing environment should support. Not all knowledge work tasks may need an environment with structure awareness at all levels. General-purpose structural computing environments, such as Callimachus [32], Themis [2], and Construct [41], are structure aware at the foundation layer (i.e., they provide their structural atom in this layer). Other more specialized environments such as FOHM [16], XCHIPS/EXTERNAL [36], and IUHM [14] are structure aware at the middleware layer, while HOSS [27] is structure aware at the operating system layer. In principle structure awareness could even be pushed below the operating system layer inside dedicated hardware – for instance in a special structural computing chip or co-processor. However, no research group has explored the latter options yet.

When designing the structural atom, both a top down and a bottom up approach can be taken. User requirements or scenarios of knowledge work drive the top down approach. With this approach the requirements at one level in the architecture (starting with the knowledge worker) is used to design the support at the level below. Following from this, the structural atom can be designed based on requirements from several different structure services. The bottom up approach is radically different. With this approach a general, flexible, and powerful structural atom is designed so that it presumably can support all possible structure services. So far general-purpose structural computing environments have tended to use the top down approach to design structure services and the bottom up approach to design their structural atom.

This current practice has led to another question. How should the gap between the top down generated structure services and the bottom up generated structural atom be bridged? The structural atom provides very general abstractions, yet the abstractions needed in the structure services are quite specific. Typically, this has been addressed in two different ways. Themis [2] and Callimachus [32] use an object-oriented approach where the structural atom can be specialized into specific structures needed in structure services. Construct [41] uses a different approach where different structure services interpret the content of the structural atom in different ways to support different structural abstractions.

A different view of what constitutes the basic building block of a structural computing environment has recently been proposed. The approach in the EAD project [20] is radically different from the structural atom approach taken by most structural computing environments. The EAD model focuses on the semantics in structuring the data. Thus, meaning is represented by structure that contains data and not (necessarily) the data themselves. The EAD model could in principle replace the structural atom in an existing structural computing environment without affecting the functionality at the upper layers in the architecture. The EAD model is said to be more powerful than the structural atom approach with respect to flexibility and its modeling capability [20].

The question of what constitutes the basic building block has puzzled the hypermedia community since the first hypermedia systems were built in the 1960s. For many years, the hypermedia community addressed the issue with mostly data oriented approaches – the WWW is a prominent example of a data oriented approach where links are embedded in the documents (nodes). Later in the evolution, links (or more generally speaking structure) were given first class status. From 1997 to around 2002, structural computing environments typically took the approach of creating a structural atom based on a data / structure synthesis view. The latest trend starting in 2003 is to try to re-raise behavior to have first class status and create a structural atom based on a data / structure / behavior synthesis view.[3] Examples of this can be found in relation to the Themis [1], Callimachus [34], and Construct [20] projects. A thorough analysis of these trends is beyond the scope of this paper.

No matter how the basic support for structure is approached, it is important to have specific goals and possible strategies in mind.

The overall goal should be to have pervasive, powerful, and easy to use basic building blocks in a structural computing environment. *Pervasive* in the sense that they can cross boundaries of platforms, operating systems, applications, etc. *Powerful* in the sense that they can model (capture) all the necessary abstractions (data, structure, and behavior abstractions according to the latest trend). *Easy to use* in the sense that developers find them easy to work with – and possibly also that they are human readable?

[3] Behavior did have first class status in some previous hypermedia systems (e.g., Multicard [31] and Proxhy [13]). However, this feature was not inherited into the modern component-based generation of (open hypermedia and structural computing) systems. Modern systems tend to provide support for behavior at a conceptual level, but few systems (if any) have implementations where behavior can be manipulated independently of data and structure.

There are several possible strategies for where to place the basic structural support (building blocks) in the computing environment. They can be provided at the operating system level for all developers to use. They can be provided in a specific programming language (say Java) for all Java developers to use. They can be provided in a dedicated framework (such as Themis) for all developers of Themis to use. They can be provided in (dedicated) hardware for all developers to use.

In summary, every structural computing system has a different approach to the structural atom. There are many issues to consider when designing a structural element – such as goal and strategy. So far no approach claims to have found anything near the perfect solution.

### 3.5 Structure Diversity or Structure Unity

Structural computing environments promote structure diversity. One of the defining factors is the support for multiple structures at the middleware and application layers. Yet, the trend is at the same time also to promote structure unity – exemplified by the existence of a single structural atom at the foundation layer in many structural computing environments.

Thinking about structure diversity versus structure unity is a useful exercise that forces developers to reconsider their design rationale. Why is it the case that many structural computing systems promote diversity at some levels and unity at other levels? Does it reflect a conscious decision? Is it a natural way to construct these environments?

Promoting structure diversity makes a lot of sense since this is exactly what is required to support knowledge work. Since knowledge workers interact with tools at the application layer, it is clearly necessary to provide structure diversity at the application layer. But, it is also necessary to promote structure diversity at the middleware layer? Structural computing researchers seem to think so, since they all promote diversity at the middleware layer in their environments.

Another question to consider is whether a unity or diversity approach to defining the structural atom should be taken? The unity approach advocates that there is only one “mother” structure (one structural atom) that everything else is based upon. The diversity approach advocates that there could be numerous different structural atoms each targeted at supporting different specialized structure services and knowledge worker tasks. The pure (extreme) unity approach could seem arrogant. Do we really think that we can build the one mother structure? The pure (extreme) diversity approach could seem impractical. Do we really want to have a structural base type for each individual application of a structure service? If the two approaches represent two extremes, then there must be several possible approaches in between the two extremes. What would they be like?

Hicks et al. [10] present a structure diversity space that can serve as a tool to study structural diversity or lack thereof in structural computing environments. The purposes of the structure diversity space are to serve as a description space in which the structural diversity of a specific computing environment can be completely and concisely described, to highlight and assist in reconciling differences in

structural diversity between computing environments, and to serve as a design space in which important diversity related decisions could be considered.

In summary, a discussion on structure diversity versus structure unity can help the community to gain a deeper understanding of possible foundations of structural computing, which will help guide future research.

## 4 Conclusion

Structural computing research has reached a level of maturity, where it is relevant and essential to define and discuss common research directions. Initially, a conceptual architecture for a structural computing environment was presented to serve as a point of reference for discussing important research issues facing the structural computing research community. The main contribution of the paper is the presentation and discussion of a research agenda for structural computing research focusing on five prominent issues: development methodology, infrastructure, interoperability, the structural atom, and structure diversity or structure unity. The structural computing field has many interesting challenges ahead. This agenda is aimed at inspiring future research in the area of structural computing.

## References

1. Anderson, K. M., Sherpa, S. A., and Van Lepthien, W. 2004. Structure and Behavior Awareness in Themis. In Proceedings of the 2003 ACM Conference on Hypertext, (Nottingham, UK, Aug.), ACM Press, 138-147.
2. Anderson, K. M., Sherpa, S. A., and Van Lepthien, W. 2003. Structural Templates and Transformations: The Themis Structural Computing Environment. In [38], 47-71.
3. Anderson, K. and Sherba, S. 2001. Using Structural Computing to Support Information Integration In [28], 151-159.
4. Anderson, K., Taylor, R., and Whitehead, E. J. 1994. Chimera: Hypertext for heterogeneous software environments. In Proceedings of the 1994 ACM Conference on Hypertext, (Edinburgh, Scotland, Sep.), ACM Press, 94-107.
5. Atzenbeck, C., Wiil, U. K., and Hicks, D. L. 2003. Toward a Structure Domain Interoperability Space. In [11], 66-71.
6. Davis, H. C., Millard, D. E., Reich, S., Bouvin, N. O., Grønbæk, K., Nürnberg, P. J., Sloth, L., Wiil, U. K., and Anderson, K. M. 1999. Interoperability between Hypermedia Systems: The Standardisation Work of the OHSWG. In Proceedings of the Tenths ACM Conference on Hypertext, (Darmstadt, Germany, Feb.), ACM Press, 201-202.
7. Ellis, C. A., Gibbs, A., and Rein, G. 1991. Groupware: Some issues and experiences. Communications of the ACM, 34(1): 38-58.
8. Grønbæk, K., and Trigg, R. 1999. From Web to Workplace – Designing Open Hypermedia Systems. MIT Press.
9. Hall, W., Davis, H., and Hutchings, G. 1996. Rethinking Hypermedia – The Microcosm Approach. Kluwer Academic Publishers.
10. Hicks, D. L., Wiil, U. K., and Nürnberg, P. J. 2004. Towards a Structure Diversity Space. In Proceedings of the 2004 ACM Conference on Hypertext, (Santa Cruz, CA, Aug.), ACM Press, 239-246.

11. Hicks, D. L. Ed. 2003. Proceedings of the Second International Metainformatics Symposium, Lecture Notes in Computer Science (LNCS 3002), Springer.
12. Jacobson, I., Booch, G., and Rumbaugh, J. 1999. The Unified Software Development Process. Addison-Wesley.
13. Kacmar, C. J., and Leggett, J. J. 1991. PROXHY: A Process-Oriented Extensible Hypertext Architecture. ACM Transactions on Information Systems, 9, 4, 399-419.
14. King, P. Nanard, M., Nanard, J., and Rossi, G. 2003. A Structural Computing Model for Dynamic Service-Based Systems. In [11], 100-118.
15. Mathiassen, M., Munk-Madsen, A., Nielsen, P. A., and Stage, J. 2000. Object-oriented Analysis & Design. Marko Publishing.
16. Millard, D. E. 2003. Discussions at the Data Border: From Generalised Hypertext to Structural Computing. In [38], 95-114.
17. Millard, D., Moreau, L., Davis, H., and Reich, S. 2000. FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability Between Hypertext Domains. In Proceedings of the 2000 ACM Hypertext Conference, (San Antonio, TX, Jun.), ACM Press, 266-267.
18. Nanard, M., Nanard, J., and King, P. 2003 IUHM, A Hypermedia-Based Model for Integrating Open Services, Data and Metadata. In Proceedings of the 2003 ACM Conference on Hypertext, (Nottingham, UK, Aug.), ACM Press, 128-137.
19. Nanard, M., and Nanard, J. 2001. Cumulating and Sharing End-Users Knowledge to Improve Video Indexing in a Video Digital Library. Proceedings of the 2001 ACM / IEEE Joint Conference on Digital Libraries, (Roanoke, VA, Jun.), ACM Press, 282-289.
20. Nürnberg, P. J., Wiil, U. K., and Hicks, D. L. 2004. Rethinking Structural Computing Infrastructures. In Proceedings of the 2004 ACM Conference on Hypertext, (Santa Cruz, CA, Aug.), ACM Press, 247-255.
21. Nürnberg, P. J., Wiil, U. K., and Hicks, D. L. 2003 A Grand Unified Theory for Structural Computing. In [11], 1-16.
22. Nürnberg, P. J. Ed. 2002. Proceedings of the First International Metainformatics Symposium, Lecture Notes in Computer Science (LNCS 2641), Springer.
23. Nürnberg, P. J. Ed. 1999. Proceedings of the First International Work on Structural Computing, Technical Report AUE-CS-99-04, Aalborg University Esbjerg, Denmark.
24. Nürnberg, P. J., Wiil, U. K., and Leggett, J. J. 1998. Structuring Facilities in Digital Libraries. In Proceedings of the Second European Conference on Digital Libraries, (Crete, Greece, Sep.), Springer, 295-313.
25. Nürnberg, P. J., Leggett, J. J., and Wiil, U. K. 1998. An Agenda for Open Hypermedia Research. In Proceedings of the 1998 ACM Conference on Hypertext, (Pittsburgh, PA, Jun.), ACM Press, 198-206.
26. Nürnberg, P. J., Leggett, J. J., and Schneider, E. R. 1997. As We Should Have Thought. In Proceedings of the 1997 ACM Hypertext Conference, (Southampton, UK, Apr.), ACM Press, 96-101.
27. Nürnberg, P. J., Leggett, J. J., Schneider, E., R., and Schnase, J. L. 1996. HOSS: A New Paradigm for Computing. In Proceedings of the 1996 ACM Hypertext Conference, (Washington, DC, Mar.), ACM Press, 194-202.
28. Reich, S., Tzagarakis, M. M., and De Bra, P. M. E. Eds. 2001. Proceedings of the Third International Workshop on Structural Computing, Lecture Notes in Computer Science (LNCS 2266), Springer.
29. Reich, S., and Anderson K. M. Eds. 2000. Proceedings of the Second International Workshop on Structural Computing, Lecture Notes in Computer Science (LNCS 1903), Springer.

30. Reich, S., Wiil, U. K., Nürnberg, P. J., Davis, H. C., Grønbæk, K., Anderson, K. M., Millard, D. E., and Haake, J. M. 1999. Addressing Interoperability in Open Hypermedia: The Design of the Open Hypermedia Protocol. Special Issue on Open Hypermedia, The New Review of Hypermedia and Multimedia (NRHM), 5, 207-248.
31. Rizk, A., and Sauter, L. 1992. Multicard: An Open Hypermedia System. In Proceedings of the 1992 ACM Hypertext Conference, (Milan, Italy, Nov.), ACM Press, 4-10.
32. Tzagarakis, M., Avramidis, D., Kyriakopoulou, M., Schraefel, M. M. C., Vaitis, M. and Christodoulakis, D. 2003. Structuring Primitives in the Callimachus Component-Based Open Hypermedia System. In [38], 139-162.
33. Tzagarakis, M., Karousos, N., Christodoulakis, D., and Reich, S. 2000. Naming as a fundamental concept of open hypermedia systems. Proceedings of the 2000 ACM Hypertext Conference, (San Antonio, TX, May), ACM Press, 103-112.
34. Vaitis, M., Tzagarakis, M., Grivas, K., and Chrysochoos, E. 2003. Some Notes on Behavior in Structural Computing. In [11], 143-149.
35. Vaitis, M., Papadopoulos, A., Tzagarakis, M., and Christodoulakis, D. 2000. Towards Structure Specification for Open Hypermedia Systems. In [29], 160-169.
36. Wang, W. 2003. Visualizing and Interacting with Hypermedia-Based Process-Centric Enterprise Models. In [38], 73-93.
37. Wang, W. and Fernandez, A. 2001. A Graphical User Interface Integrating Features from Different Hypertext Domains. In [28], 141-150.
38. Wiil, U. K., Nürnberg, P. J., and Hicks D. L. Eds. 2003. Structural Computing: Research Directions, Systems, and Issues. Special Issue on Structural Computing, Journal of Network and Computer Applications, 26, (1).
39. Wiil, U. K., Tata, S., and Hicks, D. L. 2003. Cooperation Services in the Construct Structural Computing Environment. In [38], 115-137.
40. Wiil, U. K. 2002. Lessons Learned with The Construct Development Environment. In [22], 9-17.
41. Wiil, U. K., 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, (Århus, Denmark, Aug.), ACM Press, 83-92.
42. Wiil, U. K., and Nürnberg, P. J. 1999. Evolving Hypermedia Middleware Services: Lessons and Observations. In Proceedings of the 1999 ACM Symposium on Applied Computing, (San Antonio, TX, Feb.), ACM Press, 427-436.
43. Wiil, U. K., and Leggett, J. J. 1997. Workspaces: The HyperDisco approach to Internet distribution. In Proceedings of the 1997 ACM Hypertext Conference, (Southampton, UK, Apr.), ACM Press, 13-23.
44. Wiil, U. K., Ed. 1999. Proceedings of the 5th Workshop on Open Hypermedia Systems. Technical Report CS-99-01, Aalborg University Esbjerg.
45. Wiil, U. K., Ed. 1998. Proceedings of the 4th Workshop on Open Hypermedia Systems. Technical Report CS-98-01, Aalborg University Esbjerg.
46. Wiil, U. K. Ed. 1997. Proceedings of the 3rd Workshop on Open Hypermedia Systems. Scientific Report 97-01, The Danish National Centre for IT Research.
47. Wiil, U. K., and Demeyer, S., Eds. 1996. Proceedings of the 2nd Workshop on Open Hypermedia Systems. UCI-ICS Technical Report 96-10, Department of Information and Computer Science, University of California, Irvine.
48. Wiil, U. K., and Østerbye, K., Eds. 1994. Proceedings of the ECHT '94 Workshop on Open Hypermedia Systems. Technical Report R-94-2038, Department of Computer Science, Aalborg University.

# Assessing the Impacts of Open Hypermedia Problems on Structural Computing

Nikos Karousos[1,2] and Nikos Tsirakis[1,2]

[1] Research Academic Computer Technology Institute,
15600, Rion, Greece
karousos@cti.gr
[2] Department of Computer Engineering & Informatics,
University of Patras, 26500 Rion Greece
tsirakis@ceid.upatras.gr

**Abstract.** This paper is focusing on some interesting issues about service publicity and usability while trying to move from the classic Open Hypermedia Systems (OHSs) to structural aware environments. Although the step towards structural computing provides a stable basis for new generation systems, these systems inherit from OHSs some long-period problems and aspects. Moreover, new usability problems are rising due to the increment of structural servers' complexity.

## 1 Introduction

In today's computer science community, the dominant trend is towards open systems. Open hypermedia systems (OHSs) employed a variety of techniques in order to spread the hypermedia services usability. These systems have tried to add new hypermedia functionality to the world.

Although open hypermedia systems are characterized by structures and openness they were usually supporting only navigational approach [5] and were lacking of supporting towards multiple domains with different structural abstractions. Research led to structural computing [7] in order to solve some data organization problems. Structural computing in a way generalizes the techniques used by open hypermedia [1]. The structural computing idea was the result of the philosophy: "primacy of structure over data" [10]. In general it contributes to a generalization of the hypermedia field.

Currently some interesting efforts of moving from OHS to structural aware systems have been reported [6, 11]. This step has shifted current OHSs drawbacks from OHS community to the structural computing area. These issues are critical while aiming to apply structural principles to real world application systems, due to their involvement in the structural services publicity and usability. Namely, the low publicity of OHSs, the unsatisfactory level of hypermedia service provision and the increment of structure service complexity are influencing the adoption of the structural computing theory into hypermedia systems.

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 108 – 112, 2004.

This paper presents the critical issues discussed above and suggests some actions aiming to a stable progress of structural computing research.

## 2 From Open Hypermedia Systems to Structural Aware Environments

The step from open hypermedia theory to structural computing tried to unify hypermedia domains under a common conceptual foundation. Furthermore, the notion of the structure awareness was the key point for this approach. Although structural computing research is in primary level, it still has some notable advantages [10]:

- Interoperability: Different applications can share their objects and their associations under different structural abstractions.
- Efficiency: Some intelligent operations that systems can support.
- Multiple domains: Provide co-existence of different domains under the same framework.
- Complexity: As SC has an object-oriented philosophy of computing, some of the previous problems are now easy-detected and addressed. Since the operations are implemented in an object oriented way, the complexity of the structural environments architecture is reduced.

Undoubtedly these advantages signify a promising future for structural computing but unfortunately there are also some remarkable problems that arise. Most of them derive both from the existed OHSs' problems and the structural aware environments' characteristics. An interesting user-based study about the usability of OHSs [2] reinforces some of the following issues.

### 2.1 Publicity

Open Hypermedia Systems have not yet been well-known to public and don't have a global usage [8]. There are three underlying categories that agglomerate the reasons for this:

**Research.** The Open Hypermedia Researches act in a closed research environment. Apart from the navigational point of view, other hypermedia fields (ubiquitous, taxonomic, etc) that have already been analyzed, have not yet reached a satisfactory level of publicity. Furthermore, it is difficult for new researches to contribute in the hypermedia research area due to the limited information resources.

**Systems' Promotions.** Open Hypermedia Systems are very powerful tools, but their benefits are not promoted enough to the public. Consequently, in the users and developers' area, the OHSs' products have not been presented successfully. A developer who wants to add a spatial or a taxonomic representation into his application does not know where to look for it and how he can use an available hypermedia service. In most cases he tries to implement custom solutions.

**Web Publicity.** It seems that (like some current working groups and organizations) a centralized (web based) information point, which can provide the appropriated fundamental information to the public, is still missing. Like other existing groups (World Wide Web Consortium, Internet Engineering Task Force, etc), the Open Hypermedia Working Group should try to continue (or revise) its work and publish an up-to-date web site that can inform a visitor about the Open Hypermedia Research.

### 2.2 Service Provision

Hypermedia services were always trying to reach an open set of potential users in order to increase the enrichment of the hypermedia functionality to the software systems. These efforts were not completed appropriately due to absences that derived from systems architecture, developer supporting, discovering problems and providing policies. These architectures lack of:

- developer support framework. [3]
- standardized methodologies and protocols for service provision. [4]
- add-hoc service usage implementations and web integration efforts [4].
- service discovery system [3].
- flexible pricing policy [9].
- service oriented hypermedia systems that can provide partial services instead of the complete hypermedia system [9].

### 2.3 Complexity Consequences

The architecture of hypermedia services was always complicated. The shift from data to structure problematizes models of representation [8] and results more complex structural services. Keeping the same service's API while both the architecture and the service kernel are changing is not an easy task. Consequently, although structural aware environments support multiple domains and make their parallel usage feasible, the effort for the developer to use only one service seems larger in structural environments than in classic hypermedia systems.

On the other hand, the interoperability issues among new structural environments are not fully faced yet.

## 3 Assuring a Successful Step

There is a general need for some conceptual decisions towards the difficulties that have been mentioned in the above sections. Specifically we can highlight them in the following three fields.

**Publicity.** It's a common belief that research community looks for a way to improve the promotion of their results in order to broaden their knowledge and ideas around this computer science field. As a first step to this direction, a web-based information place should be created, in order for researchers to be informed about structural computing and structural aware environments.

**Service provision.** The absence of both developer's support frameworks and the standardization of service provision techniques will raise many difficulties in development procedure, when incorporating structural computing infrastructure. There is also a necessity to implement a service discovery system and to become an inherent part of any system with structural computing philosophy. This will contribute to the widespread of the service provision. Finally, the adoption of service oriented architectures can boost the usage of structural services from users and developers.

**Complexity.** Since the research community is still in the progress of establishing an agenda for building structural computing environments, we ought to work on keeping clear structural service APIs that are easy to use. Furthermore, we should try to hide complicated structural operations in the internal of the services.

## 4 Conclusions

Structural computing is still in its early stages of research and is characterized as a revolutionary new computing paradigm. The step from the structural computing principles to the structural aware environments and applications is a very difficult task. In order this effort to avoid closed theoretical approaches or implementations of non wide acceptance systems, the drawbacks of open hypermedia systems and the structural service provision issues have to be faced. The structural computing community should take into consideration the findings of the OHSs history towards the efficient creation of the new generation systems.

## References

1. Anderson, K. M., Sherba, S. A., Lepthien, W. V. (2003). Structure and behavior awareness in themis. In Proceedings of the ACM Hypertext 2003 Conference, pp.138-147, (Nottingham, England).
2. Hicks, David L. (2002). In search of a user base: Where are the B's?. Proceedings of MetaInformatics 2002, Esbjerg, Denmark, August 8-10, 2002.
3. Karousos, N., Pandis, I. (2003). Developer Support in Open Hypermedia Systems: Towards a Hypermedia Service Discovery Mechanism. Proceeding of Metainformatics Symposium (MIS' 03), (Graz, Austria).
4. Karousos, N., Pandis, I., Siegfried, R., and 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.
5. Michail Vaitis , Manolis Tzagarakis, George Gkotsis (2004). An Engineering Perspective on Structural Computing-Developing Component-Based Open Hypermedia Systems. International Workshop on Web Engineering, ACM Hypertext 2004.
6. Millard, D. (2003). Discussions at the Data Border: From Generalised Hypertext to Structural Computing. Special Issue on Structural Computing, Journal of Network and Computer Applications, 26, 1, (January):pp. 95-114.

7. Peter. J. Nürnberg, J. J. Leggett, E. R. Schneider (1997). As we should have thought, in: Proceedings of the 1997 ACM Hypertext Conference, ACM, ACM Press, Southampton, UK, 1997, pp. 96–101.
8. Peter. J. Nürnberg, Schraefel, M. C. (2003). Relationships Among Structural Computing and Other Fields. Special Issue on Structural Computing, Journal of Network and Computer Applications, 26, 1, (January).
9. Shackelford, D. E., Smith J. B., Smith F. D. (1993). The Architecture and Implementation of a Distributed Hypermedia Storage System. In Proceedings of the 1993 ACM Hypertext Conference, (Seattle, WA, Nov), ACM press, pp. 1-13.
10. Uffe K. Wiil, Peter J. Nürnberg, David L. Hicks, (2003). Structural Computing - Research Directions Systems and Issues. Special Issue on Structural Computing, Journal of Network and Computer Applications, 26, 1, (January), 3-9.
11. Weigang Wang (2003). Co-existence and visualization of multi-domain hypertext structures. Special Issue on Structural Computing, Journal of Network and Computer Applications, 26, 1, (January).

# Structural Engineering: Processes and Tools for Developing Component-Based Open Hypermedia Systems

Michail Vaitis[1], Manolis Tzagarakis[2, 3], George Gkotsis[3], and Panagiotis Blachogeorgakopoulos[3]

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

**Abstract.** The emergence of Component-Based Open Hypermedia Systems aims at the releasing of Hypermedia and Web applications from the monocracy of link as the information structuring primitive. Instead, an open set of structure services, each one providing structure abstractions relevant to a specific domain, is offered to an open set of client applications. Nonetheless, the lack of an engineering framework guiding the development process of CB-OHS has a part in their limited exploitation. In this paper, we analyze the characteristics of CB-OHS from an engineering approach, and we propose a framework and a number of tools, supporting all phases of their development process.

## 1 Introduction

Many researches during the past decade have pointed out a certain kind of inadequacy in both Hypermedia and Web applications, concerning the information structuring abstractions they provide. This "Structure Crisis" mainly originates from the nature and implementation of the notion of link. In the Web, links are limited in functionality (they constitute starting-points for unidirectional jumps), embedded into the HTML file (so characterized as "second-class" entities). In Hypermedia Systems, although links and anchors are first class entities, they are employed for incarnating all information structuring situations. Unfortunately, any closed set of abstractions cannot be guaranteed to be useful in a practical sense for all possible applications [13]. This situation raised convenience and efficiency problems, besides lack of standards and interoperability capabilities.

A significant amount of research and development efforts aiming to overcome the above issues has resulted in the releasing of the structure abstractions from both data and core systems' functionality. Instead, structure abstractions have been promoted to first class entities, being provided to third-party client applications on demand,

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 113 – 128, 2004.

through specific software components (called *structure servers*) that form the middleware part of a tri-tier architecture (figure 1).

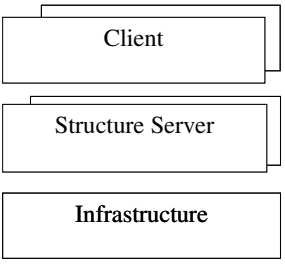



**Fig. 1.** Component-Based Open Hypermedia Systems architecture

Component-Based Open Hypermedia Systems (CB-OHS) deliver an open set of structural abstractions to an open set of client applications, while all fundamental functionalities (like persistent storage, concurrency and event notification control, versioning and naming) are provided by the infrastructure backend. The emergence of CB-OHS as the realization of structure emancipation, has further promote research in both the analysis of the different ways people use to structure information (called *hypermedia domains*), and the development of systems and tools that assist these structural activities (belonging to the "B-level" or "C-level" of work[1]). The new field of *structural computing* (SC), asserting the "primacy of structure over data" [13], is aiming to shape the theoretical and practical foundations upon which structure services will eventually become ubiquitous to all computing environments.

Although CB-OHS and structural computing are among the first forerunners of nowadays trend for service-oriented computing [17], there is little acceptance of their potential role in hypermedia and web applications development efforts [14]. We argue that one of the reasons for this situation is the lack of a predefined software engineering framework for CB-OHS, coupled with the appropriate tools to support it. The adoption of ad-hoc development methodologies drives to the production of systems that lack certain essential characteristics. The development of a structure server is a complicated task to be repeated from scratch every time a new structure abstraction has to be supported [25]. In this paper we analyze structure servers from a software engineering point of view, and propose a development framework involving all aspects of their life cycle.

The rest of the paper is organized as follows. In section 2 we present the field of structural computing and describe the functionality and internal architecture of related systems. In Section 3 we propose an engineering framework aiming to steer all the development processes of CB-OHS, while in section 4 we concentrate on the appro-

[1] According to Douglas Engelbart [5], there are three types of work that can be performed in an organization. The A-level is the work of the organization itself. The B-level is work that develops tools to improve the ability of people performing A-level work, while C-level is work that develops tools to augment the ability of people performing B-level work.

priate tools supporting the framework. Section 5 comments a number of existing structural computing environments, and finally section 6 concludes the paper and presents future research and development directions.

## 2 Structural Computing

Hypermedia has been used to support a wide variety of user tasks. These tasks range from Bush's association of information to more elaborate activities, such as hyperfiction authoring and reading, information analysis and classification. Each of the tasks exemplifies how the human mind perceives structure in different problem domains. The identification of new problem domains is the main concern of *hypermedia domain research*. On the contrary, *hypermedia system research* is focused on designing and building the computational foundations to support people working with structure, concentrating especially on issues regarding openness. The Open Hypermedia movement [16] originated from such an approach. Yet, the conceptual foundations of Open Hypermedia – the underlying structures and behaviours – have all focused on supporting one task: information navigation. As it has been shown [13], the abstractions provided by systems supporting information navigation cannot address issues in new domains (such as spatial or taxonomic) in a convenient and efficient way. These domains require structural abstractions markedly different from those used to support navigational hypermedia, manifesting, thus, a gap between hypermedia domain and system research. The need for delivering the tailored support required by different domains gave birth to Component-Based Open Hypermedia Systems (CB-OHS).

CB-OHS are the incarnation of a new approach in solving data organization problems, called *structural computing*. Research in structural computing focuses on all aspects of information structuring problems. Without being biased towards supporting only navigation among data items, structural computing attempts to provide a framework where a number of different hypermedia domains can co-exist. By providing such a unifying framework, it aims at narrowing the gap between hypermedia domain and system research. Structural computing focuses on providing general structure-oriented models and services that are able to be adapted to domain specific abstractions easily and efficiently. A 'hypermedia domain' is defined by a coherent set of abstractions that solve a particular data organization problem. CB-OHS are able to support well known domains, such as navigational, argumentation, spatial, taxonomic and configuration management, as well as other 'exotic' domains, such as workflow, hyperfiction and linguistics.

The provision of dedicated structure services for each domain, results in a more convenient and efficient utilization of its abstractions, improving in turn the performance, quality and cost-effectiveness of client applications. This fact is acknowledged as a great benefit of CB-OHS, since contemporary systems struggle with the close coupling of structure models to their infrastructure.

Structural computing systems may be viewed as part of multiple open service systems, i.e. systems supporting arbitrary middleware services that can be divided into

infrastructure and application services [25]. Structural computing attempts to change the way the invisible, but important, infrastructure of contemporary Open Hypermedia Systems work, in providing open structure-based services in heterogeneous environments. Thus, structure-based services are considered as components in the context of service-oriented computing (SOC), where services reuse and composition constitute a fundamental activity for application development [3, 17]. Therefore, structural computing focuses on the developer's side, aiming to provide tools and services for assisting the development of structure servers and client applications.

In figure 2, the conceptual internal architecture of CB-OHS is presented. The various entities of both middleware and infrastructure layers are described below, along with the appropriate protocols and interfaces.

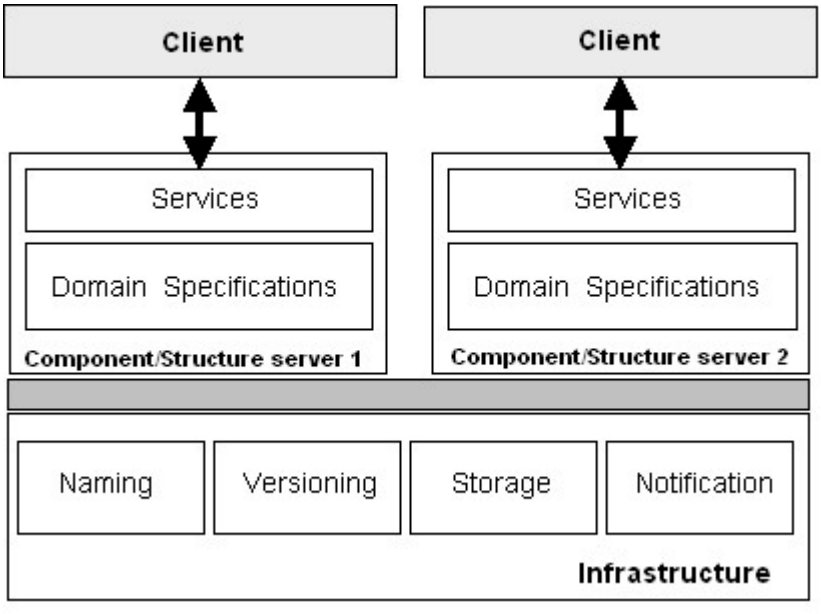



**Fig. 2.** Conceptual internal architecture of CB-OHS

- *Component/Structure server*: Reifies the domain specific abstractions, providing the domain specific services to clients. They are semi-autonomous components, since they rely on the infrastructure services for common functionality. They establish a well-defined interface for communication with client applications.
- *Domain specifications*: Comprise specifications about the structure abstractions of the domain in terms of both structure entities (patterns) and behavior semantics. Behavior models the computational aspects of the domain and can be divided into *internal operations* used mainly for consistency reasons (e.g. to

affirm conditions and constraints or to interpret abstractions in a specific manner), and *external operations* invoked by clients.

- *Services*: Implement the domain specific external operations of the domain. They are available to clients through the use of a specific interface (e.g. `openNode`, `traverseLink`).
- *Infrastructure*: Includes the fundamental functionality that is available to all structure servers. Persistent storage, naming, event notification control and versioning constitute essential common services. A well-defined protocol is provided for the communication among structure servers and infrastructure services.
- *Storage*: Provides persistent storage services for structures as well as for domain specifications. The storage protocol manipulates primitive (domain neutral) structure entities. It is the responsibility of the structure server to transform (or cast) them to the domain specific structure abstractions. Domain specifications are managed separately in a *repository*, in order to support reusability and extensibility of structure patterns among structure servers.
- *Client application*: Any third-party program that requests structure functionality from one or more structure servers. To utilize structure services, clients may be either custom-build applications, or extensions to existing applications. In the later case, either direct extensions are made, or wrapper programs are separately developed.

The presented internal architecture of CB-OHS implies the necessity of a methodology for both structure patterns and behavior semantics specification. Although such a methodology is still an open research issue in the structural computing community, the resultant benefits have already been underlined:

- Better understanding of the domain. So far, domain foundations are hard-coded into systems and services, and are informally described.
- The domain specifications could be the framework of a structure server. It may be possible to automatically configure a structure server by setting or modifying structure specifications.
- Exploitation of common structures among different domains, thus enhancing reusability and interoperability.
- Narrowing the gap between hypermedia domain and system research, by providing a common framework to express structural abstractions.

As stated above, the transition from early monolithic hypermedia systems to Open Hypermedia Systems and recently to CB-OHS was directed by the vision to provide open structure-oriented functionalities to every concerning application, in a convenient and efficient way. The layered architecture of CB-OHS aims to improve the work of both application engineers and structure server developers, enabling them to utilize high-level abstractions offered by the underneath layer.

Nonetheless, the lack of an engineering framework guiding the development process counteracts most of the anticipations of structural computing.

## 3 Structural Engineering

Hypermedia and web applications are differentiated from conventional software products in a number of characteristics, including navigability, provision of search mechanisms, appropriate content organization, aesthetic and cognitive aspects. As pointed out in [11], hypermedia applications "*uses associative relationships among information contained within multiple media data for the purpose of facilitating access to, and manipulation of, the information encapsulated by the data*", while in [4] a Web Hypermedia Application is defined as "*the structuring of an information space in concepts of nodes (chunks of information), links (relations among nodes), anchors, access structures and the delivery of this structure over the Web*". The above definitions imply a number of specific activities during hypermedia development, such as content acquisition and structuring, navigational and aesthetics design, and multimedia synchronization. The fields of Hypermedia and Web engineering have emerged, aiming to provide a systematic (scientific and practical), disciplined, quantifiable approach to the development, operation and maintenance of hypermedia (or web) applications [11, 7].

The development process of a hypermedia application includes a *design phase*, where issues like application architecture, content scope, structure, depth, granularity, presentation metaphor, viewpoints and access mechanisms are considered [11]. A number of design models have been proposed in the literature to assist this particular phase (e.g. HDM [6], RMM [8] and OOHDM [20]). What is common in all the aforementioned applications, development processes and design models, is the implied support of the navigational domain, resulting in the utilization of constructs like the node, the anchor and the link. Since structural computing perceives the navigational domain as just an instance (even the most significant) of an open set of structure services, hypermedia and web applications has the potential to exemplify customized structural abstractions according to their needs. Consequently, some modifications should be carried out in their development process.

We introduce *structural engineering* as the framework referring to a systematic and disciplined approach to the development, operation and maintenance of applications and infrastructures that solve structure-oriented problems. We argue that the design phase of hypermedia and web applications should follow or comprise a *structure assessment phase*. The purpose of this phase is to analyze the structure requirements of the application and identify the structure services that have to be used. During the implementation phase the developer should locate the appropriate structure servers and exploit their protocols. In case the application designer or developer cannot identify or locate a structure service, there is an opportunity for the establishment of a new hypermedia domain.

For the definition of a structural engineering framework, we concentrate on the special characteristics of structure servers, viewed as software components.

### 3.1 Characteristics of Structure Servers

So far, [2] is the only work that has addressed a number of engineering requirements for structure servers. We extend this list in order to provide a more complete illustration on the subject.

- *Structural completeness*: The structure abstractions supported by the structure server should completely solve the structure-oriented problems of the domain.
- *Size*: Structure servers are considered small to medium software projects. The main task is the development of the middleware component, as the infrastructure is already functional.
- *Distribution and Heterogeneity*: Structure servers should operate in a distributed environment composed of different hardware and software platforms.
- *Specifications evolution*: The decision for the construction of a new structure server should be taken only when the application needs could not be satisfied by existing services. This prerequisites a deep study of the application domain, so there is only a small possibility for the specifications to be changed during the development of the structure server. Nevertheless, the usage of CASE tools during the implementation project minimizes the cost of unexpected requirements changes.
- *Reusability and Extensibility*: Structure services at a fine granularity level constitute building blocks that have a great potential to be extended or reused during the development of other, more complicated ones.
- *Life time*: The duration of a structure server is long, presuming that the decision for its development is carefully determined. As the software implementation technologies continually evolve, structure servers may need to migrate to different platforms from time to time, while providing a constant interface to third-party applications.
- *Robustness, Scalability and Availability*: These properties are essential since structure servers may be used continually by a huge number of applications.
- *Introspection capabilities*: Structure serves should be able to communicate their behavior to other applications (i.e., their services interface and descriptions, location, and access control details).
- *Interoperability*: Structure servers that provide functionality for the same hypermedia domain should be able to interoperate. In addition, a useful requirement is the existence of supporting mechanisms for the transformation of structures between different hypermedia domains. In this way, structures may be shared among structure servers.

### 3.2 Life Cycle of Structure Servers

Although the size of the structure servers is usually small, there is a need for a disciplined development methodology, due to the demanding characteristics that should be met, as presented in the previous subsection. Based on the conventional software process phases of Specifications, Development (itemized in analysis, design, coding,

and integration), Testing, and Evolution/Maintenance [18, 19], we propose the following life cycle for the structure servers (figure 3), while in the following paragraphs we describe each one of its phases.

The proposed model condenses our experience in developing CB-OHS in the Callimachus environment [21]. It combines notions and features from a number of well-defined process models (waterfall, prototyping, and fountain), appropriately adapted to the special characteristics of structure servers. Although we have followed it in our work with satisfying results, we do not assert that it is the "only" suitable one. Instead, we propose it as a starting point for the establishment of a structure-oriented hypermedia and web applications engineering.

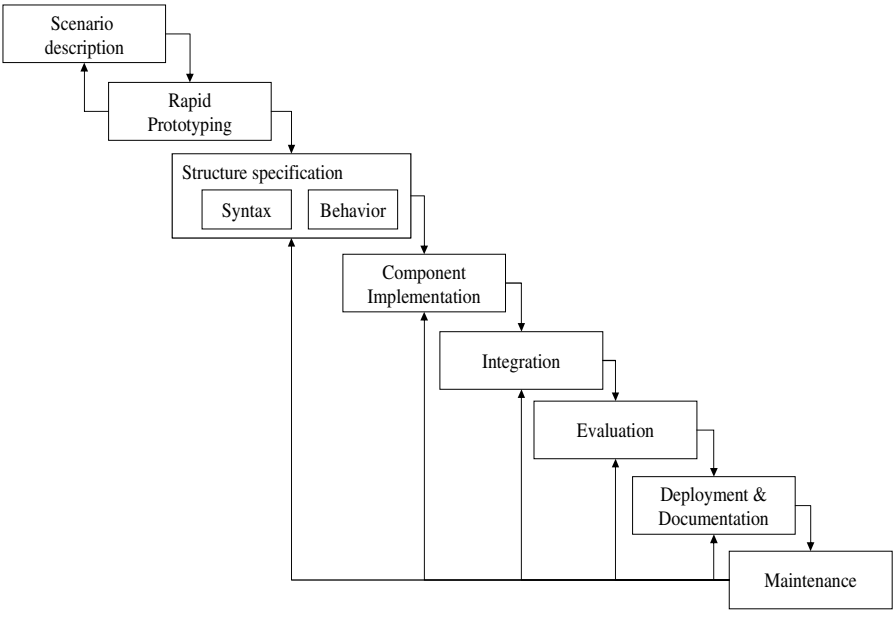



**Fig. 3.** CB-OHS life cycle

**Scenario Description**

We incorporate the scenario-based specification for open hypermedia systems [16], providing some essential modifications. All functionality proposed to be part of a given structure server should be justified through one or more scenarios of its use. That is, when a proposal that some given structure abstraction should be implemented is arisen, a scenario based on actual or foreseen use should be mapped out. This policy facilitates discussions among hypermedia application designers and developers, as to better specify the desired structure functionality and avoid "reinventing the wheel". The description of a scenario could include the following paragraphs:

- Goals (name of each goal, plus a description of it),
- Characters (the different kind of users of the service, plus a description for each one),

- Data (some examples of data items that may be associated together with the structure abstraction),
- Requirements for third-party applications (requests),
- Structure configuration (description and constraints among the structural elements),
- Behavior description (operations and propagation of them, synchronization among elements),
- Infrastructure requirements (storage, naming etc.).

**Rapid Prototyping**

The output of this phase is a prototype structure server that simulates the intended functionality, while not being fully operational in real. In this way, the evaluation of the scenario is possible and the developer of the client application has the opportunity to test the effectiveness and correctness of the desired services. Accrued ambiguities or misunderstandings are clarified and modifications to the scenario paragraphs are revealed. This testing and backtracking cycle eventually leads to a complete and correct scenario specification.

**Structure Syntax Specification**

In this phase, the *structure model* of the domain is specified. The purpose of the structure model is to make concrete the desired structure abstractions and distinguish them from the data abstractions (where they usually reside in traditional software applications). In this way, structure is elevated to a first class entity, enabling users, designers and developers to discuss and reason about it. Such a model should contain the basic structural elements, their properties and the connection constraints among them and among data items. In addition, a thorough analysis of already developed components should be carried out, in order to detect relevant elements that maybe reused. Methodologies that have been used for the specification of structure models include UML (in the Construct [27] and Themis [1] structural computing environments), XML (in the Callimachus CB-OHS [21]) or proprietary formalizations (like FOHM [12]).

**Structure Behavior Specification**

Behavior embodies the computational aspects of a domain and is tightly coupled with the structure model of it. There are two categories of behavior, depending on the calling entities: Services, that are available to clients through the server protocol, and internal operations, that are used by the structure server internally (e.g. for consistency checking), or when communicating with the infrastructure. In spite of the word “computing” in its name, the field of structural computing has no significant research results to present regarding behavior. In most cases, behavior is considered as an “add-on” that is developed on top of the structure model. Recently, structure syntax and behavior are considered by many researchers as different views of the same “whole” [23, 15], introducing the possibility of specifying syntax and behavior within the same phase or even in one step (for example, using UML in Construct [27]).

**Component Implementation**
Based on the outcomes of the previous phases, during component implementation the structure server is being reified. Activities that should be carried out are the implementation of the structure model and internal operations, the realization of the protocol (interface) for communication with the clients, the exploitation of the infrastructure services, and the eventual reuse, extension or customization of already existing components. A corner-stone task is the implementation of functions that cast the neutral structural objects stored by the infrastructure, to the specific structure elements concerning the application domain. In [21] two internal layers are identified in a structure server: The Abstraction Factory Layer (AFL) which is responsible for reifying un-typed structural objects to domain specific abstractions, and the Abstraction Utilization Layer (AUL) where the domain specific abstractions — once created — may be used by clients requesting structuring functionality.

**Integration**
Integration is a dispensable task in current structure servers, since they are developed as embedded services in the development environment and they are tightly-coupled with it. As we envision the independence of structure services from their development environments, such a phase is essential. Activities to be carried out include integration within web servers, binding of port numbers, and arrangement of authoritative and security issues.

**Evaluation**
The evaluation activities comprise mainly the ensuring of properties such as functionality, performance, compatibility, reliability, and usability.

**Deployment and Documentation**
The deployment activity turns the structure server in operational mode, so clients may use its services. A certain prerequisite for that is clients' ability to discover and locate the required service. The documentation activity includes both the registration of the structure server to the dedicated directory services, and the configuration of its introspection capabilities. In [9] a Hypermedia Service Description Language is proposed, providing a wide range of information that requesting clients can exploit, such as host and port numbers, interfaces definitions, comments, etc.

**Maintenance**
As mentioned before, if a careful analysis of the structure abstractions is carried out, the functional part of a structure server is almost unlikely to evolve. Maintenance is mainly engaged in *corrective*, *adaptive* and *perfective* tasks, targeting to correct, complete and efficient structure services.

## 4 Tools and Services

Tools that facilitate the entire development process of structure servers are still missing. In the following, we list a number of tools that seem promising in deploying the

rich services of CB-OHS and furthermore strengthen the structural computing community. These tools are classified into two categories: *Theoretical* tools, aiming at supporting problem analysis within the structural computing framework, and *development tools*, attempting to assist developers while working with CB-OHS.

### 4.1 Theoretical Tools

As already mentioned, CB-OHSs are an incarnation of structural computing, which suggests a specific approach to problems dealing with organization of data. Being a technology as well as a philosophy and school of thought, structural computing requires new ways that will help to: (a) Examine its own foundations, and (b) Analyze real-world organizational problems in a proper manner. These tools are needed primarily by analysts and to a lesser degree by developers. Theoretical tools are required on two important research fronts, analyzed below:

1. *Structural completeness* of models and systems: Since structural computing (and thus CB-OHSs) asserts that it provides a framework able to cover any need for working with structure, processes are required that will examine to what degree models may cover or solve structural problems. The coverage of these models determines their *structural completeness*.

2. Methodologies for *structural analysis* and *decidability*: CB-OHS support domains in a very abstract way. This means that the abstractions provided by components solve a *family of problems*. However, real life organizational problems, do not have an abstract view, but are very concrete. Currently there are no systematic approaches to reduce a real life organizational problem to a hypermedia domain; or more concisely, given an organizational problem, how can we determine to which hypermedia domain it belongs? Consider for example hierarchical security models that support users and groups, which in turn may consist of other groups. Is such a hierarchical security model a special case of the taxonomic domain? Does this problem introduce a new domain? Currently, structural computing cannot answer these questions in a systematic manner. In order to do so, methods for structural problem analysis need to be established, methods that can compare models and determine their differences, as well as methods that are able to decide whether or not a particular organizational problem belongs to a hypermedia domain or not. An organizational problem is said to be *decidable* within a structural computing environment if there exists an effective and systematic procedure (i.e. comprised of finite steps) that solves the problem within the structural computing framework. All those theoretic procedures determine the property of *structural decidability* of a system.

### 4.2 Development Tools

Besides the theoretical tools needed by analysts when working within a structural computing environment, actual development tools are needed by developers that will help them taking full advantage of the provided services of CB-OHS. They deal with every aspect of CB-OHS, involving model and system issues.

1. *Tools for structure syntax definition and configuration.* The development of components would be substantially easier if a specification formalism is available.
Such formalism should be open to extensions, model-neutral and provide a common ground for cooperation. Although initial attempts of such formalisms exist (e.g. [1], [27], [21]), they do not cover all domain specific aspects and have not been excessively de-ployed in order to see their shortcomings.

*2. Tools for structure behaviour specification.* While tools for structure syntax configuration are aiming primarily to syntactical aspects of structure, tools for structure behaviour specification are aiming to dynamic and computational aspects. Based on the structure model definition, tools for behaviour specification would allow controlling the life-span of structural abstractions, how structural abstractions interact and how they react to messages.

3. *Tools for discovering components in CB-OHS*. The plethora of potential components provided by CB-OHS raises issues not only relevant to components usage [10], but also to their location searching and their status and availability interrogation. While naming services can solve the problem of locating structure servers [22], this is possible only if their names are known. When names of structure servers are unknown, locating them in contemporary CB-OHS is rather impossible. The later characterizes the situation of developers that get in touch with CB-OHS for the first time. In such cases, they do not know what kind of structure servers exists, and more important, how they may deploy them. The act of attempting to locate available structure servers without prior knowledge of their existence or their name is referred to as *discovering of structure servers/components*. The tools of this category can take the form of browsers that examine the available structure servers within a CB-OHS and report on their properties, or the form of APIs that allow the integration of discovery mechanisms into third-party applications. Irrespective of the form these tools may take, special protocols are needed to facilitate the discovery of structure servers. By the term "properties of a structure server", we denote all these characteristics that distinguish one structure server from the other. These include: the name of the domain a structure server is servicing, the protocol the structure server is using to receive requests and in general to communicate, its availability, etc. Maintaining these characteristics within a special repository at the infrastructure, and selecting an appropriate representation mechanism, would also ease the development of CB-OHS clients (as well as the integration of third-party applications), since client-side code could be generated automatically. For example, using WebServices and in particular WSDL to represent the provided services, would allow the automatic generation of client-side protocol stubs. Going with these thoughts even further, this approach would also allow the runtime binding of protocol classes into clients, as it is the case on the World Wide Web.

## 5 Related Work

The questions of how to support development tasks in structural computing have already been the concern of contemporary CB-OHS. In this section a number of

existing structural computing environments are briefly presented, emphasizing on their supported development tools.

The *Themis* structural computing environment [1] consists of a framework interface, a generic structure server and two extension subsystems for the definition of structure templates and structure transformations, respectively. Application developers can extent the generic structure server by defining templates for the needed domain-specific structures. The template subsystem provides also instantiation operations for structures, which can be manipulated through the framework interface. The primary structure abstraction is the *Element* abstract class, associated with an open set of attribute-value pairs. The value of an attribute may be another element instant. Element instances belong to either of two subclasses, *Atom* and *Collection*, where the second subclass aims to group together other elements. These simple conceptual constructs can be combined to support a variety of domain-specific structures, both tree-based and non-hierarchical. The transformations subsystem provides supporting operations that automatically transform structure instances from one template to another. This functionality is delivered through custom-developed plug-in's that are loaded on demand into the transformation subsystem. Themis utilization in real application development projects has resulted in reducing the amount of code required, along with raising the level of abstraction, such that it is easier to understand and maintained.

The *Construct* structural computing environment [25, 27] is designed to support the development and hosting of an open set of structure and infrastructure services. The development process consists of using a UML tool to specify the classes (in terms of both state and behaviour) that make up a new hypermedia service. The derived diagram is automatically transformed to an IDL specification, which in turn delivered to the *Construct Service Compiler* (CSC). CSC produces a set of files, including an XML DTD, a skeleton service, and a set of common service behaviours. The skeleton service consists of a set of classes specifying a set of methods having their bodies (semantic parts) empty. The developer has to fill the missing code and load the service to the Construct environment. Generated services are available not only to third-party clients, but also to the development environment itself. The current set of services includes navigational, metadata, taxonomic, spatial, cooperation, and data mining services, while some of them are also provided on the web [24].

The *Callimachus* CB-OHS [21] provides the framework in which various hypermedia domains can co-exist. Motivated by the complexity of designing and implementing structure servers, it supports a methodology and a number of tools to facilitate these tasks. A formal XML-based syntax for the definition of domain-specific structure abstractions and their interrelationships is offered, setting up *structure templates*. *Structure types* are the building blocks for templates, while each structure type is composed of a unique id, a name, and an arbitrary number of *properties* and *endsets*. Instantiation of structure types produces *structure objects*, comprising the basic first-class entities manipulated by the system. Properties take one or more values of a specific data type, while endsets are placeholders for structure objects' ids. A set of core operations has been implemented, enabling the creation, modification, deletion, loading, saving and querying of structure objects. The methodology of developing structure server within Callimachus is divided into two phases: the *structural design*

*phase* and the *behavioural design phase*. During the former phase, the developer defines the structure template, while during the later he/she employs the set of core operations in order to build the computational aspects of the domain.

As it is evident, contemporary CB-OHS have already attempted to approach and solve development issues within their systems. Nevertheless, the approaches of current CB-OHS focus only on specific tasks e.g. formal structure specification, building structure servers  without providing an integrated and complete development environment.

## 6 Conclusions – Future Work

In this paper we attempt to approach structural computing systems from a software engineering point of view, motivated by their little acceptance by the development community. We believe that this is partly due to the lack of an engineering framework and a set of development tools, which make the use of CB-OHS cumbersome and difficult. Without proper development tools, developers hesitate to incorporate CB-OHS into their tasks. We argue that development issues should be explicitly addressed within CB-OHS and reflected by the infrastructure.

Based on well established software engineering practices and on the experience gained from previous works, we present a development process model, centralized to emphasize critical characteristics of CB-OHS, rather than exploiting specific technologies. Towards establishing proper development tools, we have identified some areas where such tools are mostly needed, including the theoretical as well as the practical aspects of structural computing. With regard to the theoretical aspects, new tools to analyze problems and evaluate the conceptual structure models are required, based on the notion of structural completeness and decidability. Regarding practical aspects, environments are needed that are able to manipulate domain specifications, including both structure syntax and behavior. Moreover, given the polymorphic nature of CB-OHS with respect to supported domains, contemporary approaches in locating structure servers seem inappropriate. New methods of discovering appropriate structure servers are necessary, directed to provide a sound foundation to enable Peer-to-Peer hypermedia.

Our current and future research plans are mainly focused on the design and implementation of the aforementioned development tools. Especially, we are working on the subject of behavior specification and the establishment of a framework for the propagation of operations through structure entities [23]. We believe that formal methods for the definition of structure semantics are a prerequisite to address the structural completeness and decidability issues.

## References

1. Anderson, K. M., Sherba, S. A., Lepthien, W. V.: Structural Templates and Transformations: The Themis Structural Computing Environment. Journal of Network and Computer Applications, 26(1), (2003) 47–71.

2. Anderson, K. M.: Software Engineering Requirements for Structural Computing. In: Proceedings of the 1st Int'l Workshop on Structural Computing (SC1, Darmstadt, Germany), Technical Report AUE-CS-99-04, Aalborg University Esbjerg, Computer Science Department, Denmark (1999) 22□26.
3. Beringer, D., Melloul, L., Wiederhold, G.: A Reuse and Composition Protocol for Services. In: Proceedings of Symposium on Software Reusability (SSR'99, Los Angeles, California, USA), (1999) 54□61.
4. Christodoulou, S., Zafiris, P., Papatheodorou, T. S.: Web Engineering: The Developers' View and a Practitioner's Approach. Web Engineering, Software Engineering and Web Application Development, Springer-Verlag LNCS 2016 (2001) 170-187.
5. Engelbart, D.: Keynote speech. 4th Int'l Workshop on Open Hypermedia Systems (OHS4, Pittsburg, PA, USA), (1998).
6. Garzotto, F., Paolini, P., Schwabe, D.: HDM – A Model-Based Approach to Hypertext Application Design. ACM Transactions on Information Systems, 11(1), (1993) 1–26.
7. Ginige, A., Murugesan, S.: Web Engineering: An Introduction. IEEE MultiMedia, 8(1), (2001) 14-18.
8. Isakowitz, T., Stohr, E. A., Balasubramanian, P.: RMM: A Methodology for Structured Hypermedia Design. Communications of the ACM, 38(8), (1995) 34–44.
9. Karousos, N., Pandis, I.: Developer Support in Open Hypermedia Systems: Towards a Hypermedia Service Discovery Mechanism. In: Proceedings of the 2nd Int'l Metainformatics Symposium (MIS'03, Graz, Austria), Springer-Verlag LNCS 2994 (2004) 89–99.
10. Karousos, N., Pandis, I., Reich, S., Tzagarakis, M.: Offering Open Hypermedia Services to the WWW: A Step-by-Step Approach for Developers. In: Proceedings of 12th Int'l World Wide Web Conference (WWW 2003, Budapest, Hungary), (2003) 482□489.
11. Lowe, D., Hall, W.: Hypermedia and the Web: An Engineering Approach. Wiley (1999).
12. Millard, D. E., Moreau, L., Davis, H. C., Reich, S.: FOHM: A Fundamental Open Hypertext Model for Investigat-ing Interoperability between Hypertext Domains. In: Proceedings of 11th ACM Int'l Conference on Hypertext and Hypermedia (Hypertext '00, San Antonio, Texas, USA), (2000) 93–102.
13. 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).
14. Nürnberg, P. J., Schraefel, M. C.: Relationships among Structural Computing and Other Fields. Journal of Network and Computer Applications, 26(1), (2003) 11□26.
15. Nürnberg, P., Wiil, U. K., Hicks, D. L.: A Grand Unified Theory for Structural Computing. In: Proceedings of the 2nd Int'l Metainformatics Symposium (MIS'03, Graz, Austria), Springer-Verlag LNCS 2994 (2004) 1–16.
16. Open Hypermedia Systems Working Group (OHSWG). http://www.csdl.tamu.edu/ohs/
17. Papazoglou, M. P., Georgakopoulos, D. (eds.): Service-oriented Computing. Communications of the ACM, 46(10), (2003).
18. Pfleeger, S. L.: Software Engineering: Theory and Practice. Prentice Hall (2001).
19. Pressman, R. S.: Software Engineering - A Practitioner's Approach. McGraw – Hill. Fourth Edition (1997).
20. Schwabe, D., Rossi, G., Barbosa, S. D. J.: Systematic Hypermedia Application Design with OOHDM. In: Proceedings of 7th ACM Int'l Conference on Hypertext (Hypertext '96, Bethesda, Maryland, USA), (1996) 116–128.
21. 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.

22. Tzagarakis, M., Karousos, N., Christodoulakis, D., Reich, S.: Naming as a Fundamental Concept of Open Hyper-media Systems. In: Proceedings of 11th ACM Int'l Conference on Hypertext and Hypermedia (Hypertext '00, San Antonio, Texas, USA), (2000) 103– 112.
23. Vaitis, M., Tzagarakis, M., Grivas, K., Chrysochoos, E.: Some Notes on Behavior in Structural Computing. In: Proceedings of the 2nd Int'l Metainformatics Symposium (MIS'03, Graz, Austria), Springer-Verlag LNCS 2994 (2004) 143□149.
24. Wiil, U. K., Hicks, D. L.: Providing Structural Computing Services on the World Wide Web. In: Proceedings of the 3rd Int'l Workshop on Structural Computing (SC3, Aarhus, Denmark), Springer Verlag LNCS 2266 (2002) 160–171.
25. Wiil, U. K., Nürnberg, P. J., Hicks, D. L., Reich, S.: A Development Environment for Building Component-Based Open Hypermedia Systems. In: Proceedings of 11th ACM Int'l Conference on Hypertext and Hypermedia (Hypertext '00, San Antonio, Texas, USA), (2000) 266 – 267.
26. Wiil, U. K.: Multiple Open Services in a Structural Computing Environment. In: Proceedings of the 1st Int'l Workshop on Structural Computing (SC1, Darmstadt, Germany), Technical Report AUE-CS-99-04, Aalborg University Esbjerg, Computer Science Department, Denmark (1999) 34□39.
27. Wiil, U. K.: Using the Construct Development Environment to Generate a File-Based Hypermedia Storage Service. In: Proceedings of the 2nd Int'l Workshop on Structural Computing (SC2, San Antonio, Texas, USA), Springer Verlag LNCS 1903 (2000) 147□159.

# A Semantic Representation for Domain-Specific Patterns

Susana Montero, Paloma Díaz, and Ignacio Aedo

Laboratorio DEI. Dpto. de Informática,
Universidad Carlos III de Madrid,
Avda. de la Universidad 30. 28911 Leganés, Spain
{smontero, pdp}@inf.uc3m.es, aedo@ia.uc3m.es
http://www.dei.inf.uc3m.es

**Abstract.** Design patterns are a valuable mechanism to capture and disseminate best practice in software design. The oft-cited definition of an Alexandrian pattern, *"a solution to a problem in a context"*, stimulates the definition of patterns from knowledge and expertise in any domain. Indeed, their application has spread from the object-oriented community, who first adopted them, through different software areas including human-computer interaction, virtual environments, ubiquitous computing, hypermedia and web engineering. This kind of patterns that describe successful solutions to recurring design problems in terms of a specific domain of application are known as domain-specific patterns.

The increasing number of available design patterns is making difficult to find the most appropriate one given a specific problem since this task requires mastery on existing design patterns. Hence, there is a need to introduce a formalism to describe them accurately and to allow a rigorous reasoning process to assist users to retrieve those patterns that solve their problems. With this purpose, we propose a semantic representation for domain-specific patterns based on the domain knowledge for which they were written and for which an ontology-based approach is applied. This representation is used as an underlying armature for complementing the informal textual pattern description by means of semantic annotations. The combination of the literary pattern representation with its formal representation counterpart could assist an intelligent search engine that supports users not just for retrieval purposes but also for the discovery useful design solutions improving, therefore, their ability to develop quality software.

## 1 Introduction and Motivation

In 1977, the architect Christopher Alexander and his colleagues introduced the *pattern* metaphor as a mechanism to capture the essence of successful solutions to recurring design problems in urban architecture [2]. Patterns are generally intended for human/manual use as they are structured but informal documents made up of a series of fields, including the problem that the pattern addresses, the context in which it is used, and the suggested solution written in natural language. This idea of capturing knowledge into patterns was adopted by the object-oriented community as an effective mean for reusing best practice in software design [10]. Moreover, their continued usage

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 129–140, 2004.

among designers and developers has turn the names of patterns into a shared vocabulary for expressing and communicating knowledge.

The oft-cited definition of an Alexandrian pattern, *"a solution to a problem in a context"*, is so general that allows patterns to be formed from knowledge and expertise in any domain. This quality together with the aforementioned benefits have spread the adoption of patterns to other fields, whether or not software-related. Some of these fields related to the software area include human computer interaction [5], security [1] and hypermedia [18]. All these kinds of patterns are embraced in a broader term denominated *Domain-Specific Pattern* and it is this category that strongly motivates our work.

In recent years, the number of domain-specific patterns has increased drastically. For this reason, repositories, tools and methods for searching and selecting a particular pattern in *an intelligent way* are required to improve the ability of developers to produce high quality software reusing good design solutions. Current approaches to represent patterns are document-based, at best supported with hypertext tools on the Web, meaning that effective use is difficult. For a designer or a domain expert, finding suitable patterns for a particular purpose is increasingly problematic. They need to be aware of existing patterns and to understand when they should be applied.

In this paper, we propose a semantic representation for domain specific patterns based on the domain knowledge for which they were written. For that purpose, both the pattern format and the domain are reflected separately in a representational vocabulary using an ontology-based approach. Subsequently, this representation is used as an underlying armature for complementing the informal textual representation of patterns by means of semantic annotations. The combination of both makes patterns mutually understandable to stakeholders and machines. Finally, this general model is applicable to domain-specific patterns whether or not software-related, providing the semantic foundation on which to build standardised patterns collected in repositories and to develop tools for intelligent reasoning. This kind of efforts comprise what Engelbart identified as B-level activities [8], that are those oriented towards reducing the product-cycle time, that is, towards improving the ability to perform our core activity. The approach presented here is intended to facilitate software reuse and, thus, to improve our ability to develop quality software by reusing tested knowledge and expertise. As the specific domain area used to illustrate this approach is the hypermedia arena, this work is expected to constitute one contribution to the B's David L. Hicks was looking for in [15].

This paper is structureed as follows: Section 2 briefly introduces the concept of Domain Specific Pattern and provides some examples of these patterns. Section 3 presents related work on the formal specification of patterns. Section 4 describes our model for the representation of domain-specific patterns from their semantics, its underlying rationale as well as their formal components. Section 5 subsequently illustrates our approach with a case study and section 6 finally presents some conclusions and the future directions of our research shall take.

## 2 Domain-Specific Patterns

Design patterns have proven that they are a natural way for the formulation of accumulated expertise. It all began in the field of building architecture. The software en-

gineering community enthusiastically embraced the patterns concept and their field of application has been extremely broadened. We can find patterns in other software areas such as hypermedia, security, user interfaces, real-time systems or even in knowledge areas such as communication, economy or education. As a consequence, large knowledge repositories of domain-specific patterns are being created. To illustrate this situation, we present some software domains that have an important literature together with an example of pattern.

- *Human-computer interaction*. These patterns are focused on solutions to problems that end-users have when they interact with systems, being the usability the essential design quality. Interaction design patterns collections are publicly available in books [5] or online [21] with a total count of more than 250 published patterns. For example, the *Progress* pattern should be used when the user wants to know whether or not the operation is still being performed as well as how much longer the user will need to wait. This pattern proposes that the application provides a valid indication of progress while it is still working [21].
- *Security*. A security pattern is a well-understood solution to a recurring information security problem the aim of which is to allow developers to make informed trade-off decisions between security and other goals. These patterns include different aspects of security such as security models, application security or cryptography [1]. Moreover, the pattern template [10] has been altered to convey more security-relevant information [16]. An example of security pattern is the *Check Point* pattern which proposes a structure that checks incoming requests, and in case of violations, the Check Point is responsible for taking appropriate countermeasures [16].
- *Hypermedia*. Hypermedia patterns are commonly used in any type of hypermedia design, including web applications, to deal with the different aspects of conceptual design (interface, content structure, navigation, dynamics, interaction). The first hypermedia patterns were presented by Rossi [18], and many more have since been published [12]. As example of hypermedia pattern, the *Index Navigation* pattern provides fast access to a group of nodes, adding a node which contains an entry point for each node of the group and vice versa [11].

Next section discusses existing representations of patterns and their suitability for capturing the knowledge domain of these patterns and the automatic support for organisation, retrieval and provision of explanations.

## 3 Related Work

Design patterns usually are expressed in natural language, through a specific structure. Although, there are different formats [17], they usually contain certain essential elements that describe *the context* in which the problem occurs, *the intent and statement* of the problem, a description of *the relevant forces* that justify why the pattern and implementations should be used and how to generate *the solution*. Although this representation is the most suitable to understand the rationale behind a pattern, there is a limitation to precision due to the use of natural language and the semantic ambiguity.

This feature does not allow any level of automation to resolve problems widely recognised such as finding and selecting the appropriate patterns or applying them.

In order to resolve these problems there have been several attempts to provide design patterns with certain formalism in the object oriented community, but not in other domains. For example, the Cornils and Hedin [6] model design patterns using reference attributed grammars with syntactic and context-sensitive rules. Eden et al. [7] define LePlus (LanguagE for Patterns Uniform Specification) from Higher-Order logic to represent design patterns as logic formulas which consist of participants (i.e. classes, functions or hierarchies) and relations imposed amongst them. Smith and Stotts [19] use an extension of sigma calculus which defines relationships between the elements of object oriented language to express design patterns. Finally, Taibi and Ling Ngo [20] are concerned with specifying both the structural and behavioural aspect of design patterns using first order logic for the former and temporal logic for the later. This approach describes patterns in terms of entities (classes, attributes, methods, objects and untyped values) and relations that express the way entities collaborate.

The main advantage of these approaches is their rigourous reasoning since they are based on formal specification languages. However, this advantage can turn into a strong disadvantage if the user cannot easily understand how patterns have been specified or how to specify new patterns. Most of the reported approaches adopt a well-defined mathematical semantics to describe the object-oriented domain that it is not readable by people from outside the software engineering profession and lacking mathematical skills. In the case of domain specific patterns, they are used by both domain experts and novices whose knowledge is focused solely on the application domain. Moreover, most of domain-specific patterns not described in terms of classes, objects, etc., but in terms of components of the application domain, so the aforementioned approaches can not be applied directly.

Finally, they reflect only the pattern solution, but a pattern is much more than this. Besides teaching the rationale behind the solution and its trade-offs [3], a pattern captures how and when to apply the solution. All of these elements are needed for the development of support tools for organisation, retrieval and provision of explanations. For example, including *the intent section* of a pattern description in the formalisation would allow for indexing of patterns according to the problem that resolve, helping users find the correct pattern for a design problem.

A formalisation of a pattern description cannot replace the understandability of the natural language description but can be thought of as a complement.

## 4 An Ontology-Based Approach

As aforementioned, the objective of our approach is to provide an underlying formalism that complements the textual description of the domain specific patterns based on the same terms experts use to describe their domain knowledge. To achieve this aim, our approach is based on the use of ontologies. Ontologies have as main motivation help programs and humans share and reuse knowledge bodies [14] providing the following benefits to our solution:

– **Well-defined semantics.** Patterns are described in textual form which can be ambiguous and sometime misleading in their understanding and application. This approach provides a formalism that allow us to know precisely what we mean when we use domain-specific terms and concepts.
– **Rigourous reasoning.** Patterns to be understood by machines need a formalism that supports reasoning. Ontologies can provide reasoning service over information expressed since ontology specification languages are usually based on a particular kind of formal logic [13].
– **Browsing/Searching.** Due to the large amount of patterns there is a need for tools to make patterns accessible in more than one way. Our approach can assist an intelligent search engine that supports users not just for retrieval but for discovery. These tools can generate different views or offer dedicated search functionality to select appropriate patterns.
– **Interoperability.** Different repositories that hold patterns formalised with different ontologies should coexist. Ontologies enable the integration of information from diverse resources, allowing different ontologies to model the same concepts of the application domain in different ways.

### 4.1 A Representation Framework

Our approach is based on three types of ontologies that supply the basic structure or armature around which domain-specific patterns can be built, as shown in Figure 1.

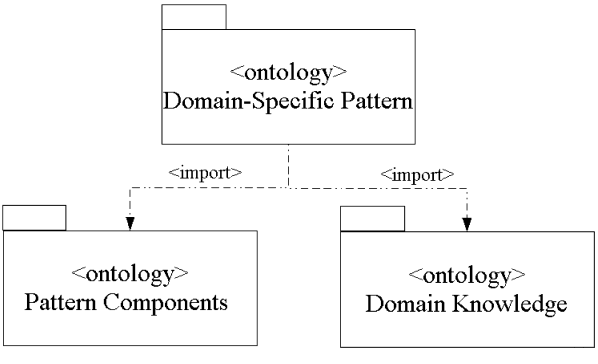



**Fig. 1.** Import relations between the ontologies for domain specific patterns using the UML package notation

– *The pattern components*. This ontology gathers the elements that are part of the pattern template, but independent of any particular domain. Patterns, in general, follow a format to describe a proven solution to a recurring design problem [2]. Depending on the level of detail of each domain more specific components to describe its patterns including diagrams, examples of use or pointers to related patterns can be added here [17].

- *The domain knowledge*. This domain ontology describes the vocabulary and background knowledge of the domain of application. Examples of this ontology can be the core knowledge needed to describe human computer interaction, security or hypermedia patterns.
- *The domain specific pattern*. This ontology imports the above ontologies to define an explicit mapping between them. This allows us to keep the pattern template and the domain specific knowledge separate since each domain depending on its needs uses more or less fields to describe its patterns and the later ontology changes according to the domain knowledge that the pattern captures.

Each of these ontologies specifies the representational vocabulary in the corresponding domain, with agreed-on definitions of the terms in declarative form. Definitions may include axioms, constraints, relationships among concepts, and hierarchies of the problem domain. Once the *domain specific pattern ontology* is specified, it acts as a semantic backbone for making semantic annotations to the textual pattern description. This framework can be instantiated in different domains for modelling their domain-specific patterns.

### 4.2 AnnotPat: An Ontology-Based Pattern Annotation Tool

Semantic annotation deals with assigning links to the semantic description of raw data in order to produce data that are machine-processable. Semantic annotation is applicable to any sort of text, audio, video, and currently to web pages in order to build the semantic web [4]. In this context, semantic annotation is formalised from two sets of objects, documents and formal representations and two functions can be created: a function from document to formal representations, called annotation and a function from formal representations to documents called index [9]. The work presented here is related to the first function.

As mentioned previously, we aim to enrich the textual pattern description to enable the development of tools for the automatic organisation, retrieval and explanation of domain-specific pattern collections.

The overall approach is depicted in Figure 2. Domain-specific patterns are annotated by domain experts or novices using an annotation tool called AnnotPat. AnnotPat provides the facility to annotate a textual pattern description with its appropriate semantic terms. Thus, a formalised pattern can be extracted and stored together with its textual representation, suitable for further processing. From the knowledge base and a inference engine different tools can be developed, such as a query tool to search and retrieve suitable patterns in terms of the ontology, a web community portal through which pattern knowledge can be gathered, stored, secured and accessed by members, or support tools that interact with other systems, to help users select and apply patterns according to the application context.

With regard to the AnnotPat tool, it reads an OWL file containing a domain-specific pattern ontology specification. OWL [1] is the W3C recommended language to describe

[1] http://www.w3.org/2004/OWL/

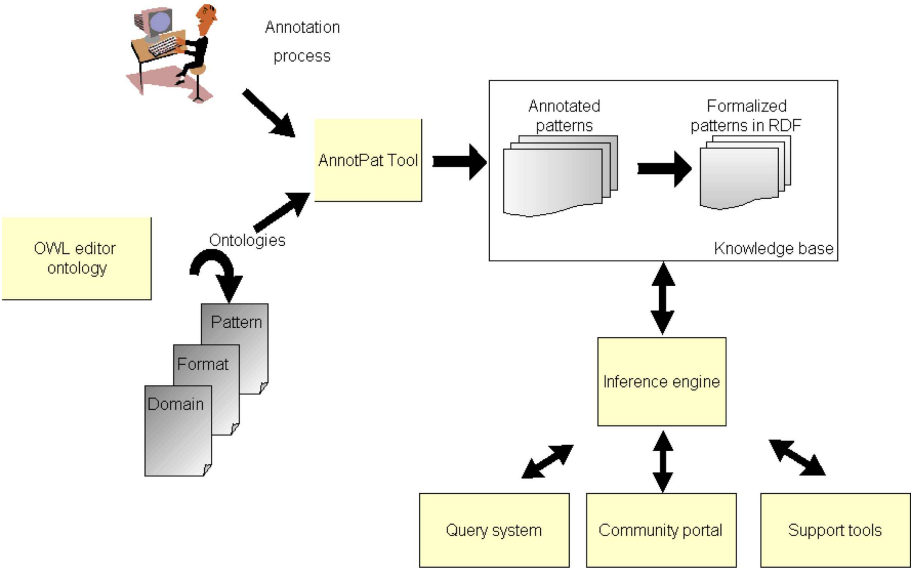



**Fig. 2.** The approach overview

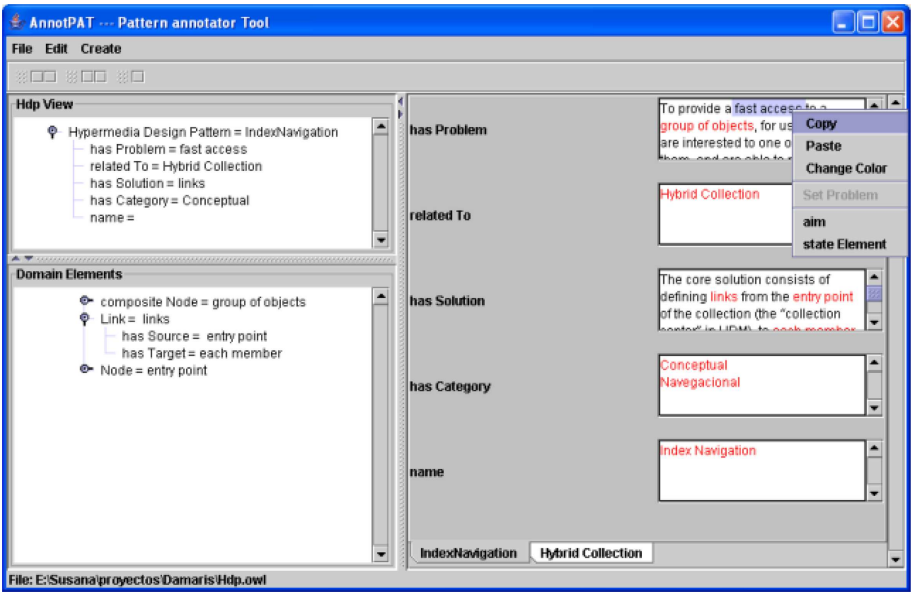



**Fig. 3.** Snapshot of the annotation interface. The user has highlighted a piece of text to make an annotation

ontologies. From this specification, the tool generates a user interface for patterns annotation. Figure 3 shows a snapshot of the annotation interface which is divided into two sides. The left side shows the status of annotations made by the user as a domain-specific pattern ontology instance. The right side shows the template pattern where the textual descriptions are typed and annotations are made. After typing the pattern, the user can make annotations in order to link words with terms. For that, she first needs to highlight a relevant piece of text and does click with the right mouse button. A pop-up menu is displayed with the possible terms according to the pattern element selected, for easier use.

In the next section, we put the high-level rationale outlined above into a concrete perspective, hypermedia design patterns.

## 5 A Study Case: Hypermedia Patterns

Before making any annotation on hypermedia patterns, we need to define the domain ontology from which these patterns capture the domain solutions. Hypermedia patterns are described by a number of terms and concepts that describe the general features of hypermedia applications. Taking into account only the elements related to their structure, all hypermedia applications can be described in a simple way from the following terms:

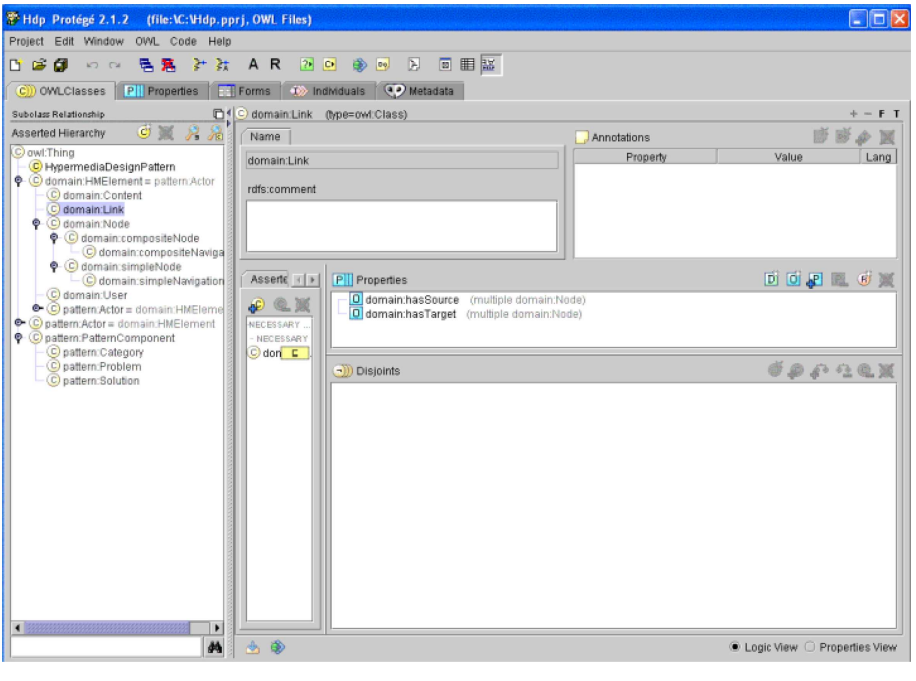



**Fig. 4.** Snapshot of the Protégé ontology editor v. 2.1.2 showing part of the domain-specific pattern ontology

```
<rdf:RDF
  ...>
<hdp:HypermediaDesignPattern rdf:ID="IndexNavigation">
  <hdp:hasSolution rdf:resource="#links"/>
  <hdp:hasCategory rdf:resource="#Conceptual"/>
  <hdp:hasProblem rdf:resource="#fast access"/>
  <hdp:relatedTo rdf:resource="Hybrid Collection#Hybrid Collection"/>
  <hdp:name>Index Navigation</hdp:name>
 </hdp:HypermediaDesignPattern>
<pattern:Problem rdf:about="#fast access">
  <pattern:state>
   <domain:compositeNode rdf:about="#group of objects"/>
  </pattern:state>
  <pattern:aim> fast access </pattern:aim>
 </pattern:Problem>
 <pattern:Solution rdf:ID="links">
  <pattern:state>
   <domain:Link rdf:about="# links">
    <domain:hasTarget rdf:resource="#group of objects"/>
    <domain:hasSource>
     <domain:Node rdf:about="#entry point"/>
    </domain:hasSource>
   </domain:Link>
  </pattern:state>
  <pattern:state rdf:resource="#entry point"/>
 </pattern:Solution>
 <pattern:Category rdf:ID="Conceptual">
  <pattern:level>Conceptual</pattern:level>
  <pattern:scope>Navegacional </pattern:scope>
 </pattern:Category>
</rdf:RDF>
```

**Fig. 5.** The *Index Navigation* pattern in RDF format

- **Node** is an information holder able to contain a number of contents. Examples of nodes are a web page, a frame or a pop-up window.
- **Content** is a piece of information of any type like text, audio, graphics, animations or programs.
- **Link** is a connection among two or more nodes or contents. A link is defined between two set of anchors.
- **Anchor** is the source or target of a link, and determines a reference locus into a node and/or content.

Secondly, the template of the domain specific pattern is defined by means of the mapping between the hypermedia domain ontology and the fields needed to describe the domain specific patterns giving *the Hypermedia Design Pattern ontology* as a result. We specified all ontologies in OWL using the Protégé [2] ontology editor and its OWL plug-in. This editor allows users to build ontologies in a frame-like way by means of classes, properties and axioms.

[2] http://protege.stanford.edu/

Finally, we can use the AnnotPat tool in order to annotate a pattern with the final domain specific pattern ontology. To demonstrate this process, we used the *Index Navigation* pattern explained in the section 2.

From the typed text of this pattern, the following annotations are made. The *name* is *IndexNavigation* and the pattern is *related to* another pattern called *Hybrid Collection* [11]. The element Category describes the *level* of abstraction and design aspect which suggest when a pattern could be used. For this example, the pattern is considered *conceptual* since no detail about the solution implementation is provided and it belongs to the *navigational* category since its aim is to organise the navigation through the hypermedia application. The element Problem describes the scenario in which the pattern is applied by means of *aim*, which uses keywords to identify the reason why the pattern should be used, and *state*, which defines what hypermedia elements participate in the problem scenario. In the case of this pattern, its aim is to provide *fast access* to *nodes*. Finally, the element Solution describes how to obtain the desired outcome by means of *state* which defines what hypermedia elements participate in the solution scenario. As the solution of this pattern is to add a new node that contains entry points for each node, the actors that participate in the solution state are: the nodes of the problem state, the new node that represents an index, and the links between the two types of nodes.

The result of the annotation process is showed in Figure 3 where the highlighted text means the annotated words. Figure 5 shows the *Index Navigation* pattern formalised as an instance of the domain specific pattern ontology in RDF [3] format ready to be shared by repositories and processed by reasoning tools.

## 6 Conclusions and Future Work

In this paper, we presented an ontology-based approach that allows designers and domain experts to enrich their domain-specific patterns with semantic annotations using their domain concepts and terms. The representation framework keeps the pattern template and the domain specific knowledge separate since each domain depending on its needs uses more or less fields to describe its patterns and the later ontology changes according to the domain knowledge that the pattern captures. This feature allow the framework to be applicable to any domain providing the formal foundation to build standardised patterns collected in repositories and *intelligent tools* for their automatic organisation, retrieval and discovery.

The most valuable contribution of this approach is to be both human and machine readable, since the text pattern description is linked to its formal representation, protecting the essence of design patterns: *express and communicate successful solutions to common problems between practitioners*. In this way, patterns become more easy to reuse and, therefore, the developer's ability to produce quality software can be improved. In a general sense, this work address a B-level activity within the Engelbart's ABC of improvement infrastructure [8] that is aimed at helping software developers to enhance their core activity.

Finally, we are working on enhancing the AnnotPat tool and provide personal and shared spaces such as web repositories or communities.

[3] The W3C recommendation for representing metadata, http://www.w3.org/RDF/

## Acknowledgements

Thanks to Damaris Fuentes, who implemented the AnnotPat tool prototype. The authors also wish to acknowledge the constructive suggestions provided by the reviewers and the research assistants. Finally, this work is supported by Dirección General de Investigación de la Comunidad Autónoma de Madrid and Fondo Social Europeo (CAM and FSE) (07T/0024/2003 1).

## References

1. *Security Engineering With Patterns: Origins, Theoretical Models, and New Applications*. Springer-Verlag, 2003.
2. C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, 1977.
3. B. Appleton. Patterns and software: Essential concepts and terminology, 2003.
4. T. Berners-Lee. Semantic web road map. w3c design issues.
5. J. O. Borchers. *A Pattern Approach to Interaction Design*. John Wiley & Sons, 2001.
6. A. Cornils and G. Hedin. Tool support for design patterns based on reference attribute grammars. In *Proc. of WAGA'00, Ponte de Lima, Portugal*, 2000.
7. A. H. Eden, A. Yehudai, and J. Gil. Precise specification and automatic application of design patterns. In *Proc. of International Conference on Automated Software Engineering (ASE '97)*, pages 143–152, Lake Tahoe, CA, USA, 1997.
8. Douglas C. Engelbart. Augmenting Human Intellect: A Conceptual Framework. Technical report, Air Force Office of Scientific Research, 1962.
9. J. Euzenat. Eight questions about semantic web annotations. *IEEE INTELLIGENT SYSTEMS*, 17(2):55–62, 2002.
10. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
11. F. Garzotto, P. Paolini, D. Bolchini, and S. Valenti. Modeling-by-Patterns of web applications. In *Advances in Conceptual Modeling: ER '99 Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling*, pages 293–306, 1999.
12. D. German and D. Cowan. Towards a unified catalog of hypermedia design patterns. In *Proceedings of 33rd Hawaii International Conference on System Sciences*, Maui, Hawaii, 2000.
13. A. Gómez-Pérez and O. Corcho. Ontology specification languages for the semantic web. *IEEE Intelligent Systems*, 17(1):54–60, 2002.
14. T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.
15. D.L. Hicks. In search of a user base: where are the b's. In *Metainformatics 2002*, volume 2461 of *LNCS*, pages 112–117, 2003.
16. S. Konrad, B. H. C. Cheng, L. A. Campbell, and R. Wassermann. Using security patterns to model and analyze security requirements. In *Proc. of the Requirements for High Assurance Systems Workshop (RHAS03) as part of the IEEE Joint International Conference on Requirements Engineering (RE03)*, Monterey Bay, CA, USA, September 2003.
17. L. Rising. Pattern forms. In *Proceedings of Viking PLOP 2003*, 2003.
18. G. Rossi, A. Garrido, and S. Carvalho. *Design Patterns for Object-Oriented Hypermedia Applications. Pattern Languages of Programs II*. Addison-Wesley, 1996.

19. J. Smith and D. Stotts. Elemental design patterns: A link between architecture and object semantics. Technical Report Technical Report TR02-011, Univ. of North Carolina at Chapel Hill, March 2002.
20. T. Taibi and D. C. Ling Ngo. Formal specification of design patterns - a balanced approach. *Journal of Object Technology*, 2003.
21. M. Van Welie. Amsterdan collection of patterns. URL: http://www.welie.com. Visited January 2004.

# Describing Use Cases with Activity Charts

Jesús M. Almendros-Jiménez and Luis Iribarne

Dpto. de Lenguajes y Computación, Universidad de Almería, Spain
{jalmen, liribarne}@ual.es

**Abstract.** The Model-Driven Development (MDD) describes and maintains models of the system under development. The Unified Modeling Language (UML) supports a set of semantics and notation that addresses all scales of architectural complexity by using a MDD perspective. Use Cases and Activity Charts are two modeling techniques of the UML. The first one helps the designers to identify the requirements of the system discovering its high level functionality. The second one helps them to specify the internal behaviour of a certain entity or subsystem of the software developed, such as a database, a graphical interface, a software component, or any specific software. However, there is not a direct way to relate/model the requirements (use cases) with their internal behavior (activity charts). In this paper we present a method for describing use cases with activity charts. Our technique also allow us to identify the two main use case relationships —include and generalization— by means of activity charts. As a case study, we will show how to use the activity charts to describe graphical user interfaces (GUI) from use cases. In particular, we will show an Internet book shopping system example.

## 1 Introduction

The general processes applied in the development of systems handle the use of solutions based on spiral methodologies [Boe88, Nus01]. These solutions are focused on an iterative use of practices of analysis and design (A&D) and the building of rapid prototypes of several parts (i.e., the business, data and presentation logic) and stages (analysis, design and coding) of the system.

In general, a development process begins with the high level elicitation and description of the requirements of the domain to be modeled. These requirements are then systematically and progressively refined under development. In the process, rapid prototypes of models of the system are simultaneously developed, which are continuously revised by the designers. These revisions change the state of the collection of requirements, modifying or removing the requirements and/or dependencies between requirements, or adding some news one. The spiral model allows us to model the presentation, data and business logic in a concurrent and iterative way. Nevertheless, although this concurrent development method does not imply an independent development between its parts —since there exist a connection— it gets the development of rapid prototypes of models, driving it on the bit parts of the system. The development of

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 141–159, 2004.

these rapid prototypes may involve different platforms (such as J2EE or .NET), domains (such as real-time systems, database, graphical interfaces, or software components) and tools (such as IDL, XML or OCL). As a result of this, the *Model-Driven Development* style (MDD) [OMG03] is generating increasing interest due to the needs of methodologies getting a rapid development and a direct connection between models.

The *Unified Modeling Language* (UML) [OMG03] is an A&D feature that supports a set of semantics and notation to address all scales of architectural complexity under MDD perspectives. For instance, the *Use Cases* and *Activity Charts* are two modeling techniques of the UML. The first one helps the designers to identify the requirements of the system, discovering its high level functionality. The second one helps them to specify the internal behaviour of a certain entity or subsystem of the software developed, such as a database, a graphical interface, a software component, or any specific software. However, a direct way to relate/model those requirements identified in the use cases with their internal behaviors modeled in the activity charts is not immediate.

In order to address this gap between models, this paper presents a method that proposes how to describe use cases into activity charts. This method can be considered as a concrete proposal to describe/connect models in the MDD arena, use cases and activity charts in this case. The technique also allows us to identify the two main use case relationships (*include* and *generalization*) by means also of activity charts. In order to clarify the method, this paper presents a case study that puts into practice our approach. This case study handles the use case model for designing *Graphical User Interfaces* (GUI). In particular, we will show a design example of an Internet book shopping system. However, this is just a case study of our approach which is enough general to be used for specifying other system views from the use case model, such as data and business logic, which can complete the system's views following the UML philosophy.

The rest of the paper is organized as follows. Section 2 includes background information on what drove the requirements and the design rationale in the Unified Modeling Language (UML). Section 3 describes a general method for describing use cases with activity charts, and the identification of use case relationships. Section 4 describes an specialization of our method oriented to the design of GUI. Then, Section 5 presents an Internet Book Shopping example that illustrates our method. Finally, Section 6 discusses some conclusions and future work.

## 2 Discussion of the Unified Modeling Language (UML)

The UML helps the designers working on analysis and design (A&D) with a consistent language for *specifying*, *visualizing*, *constructing*, and *documenting* the artifacts of software systems. One of the primary goals of the UML is to enable *meaningful exchange* of model information between tools, agreement on semantics and notations (for instance, providing IDL specifications as a mech-

anism of model interchange between OA&D tools). The choice of what models and diagrams to create has a profound influence upon how a problem is attacked and how a corresponding solution is shaped. Every complex system is best approached through a small set of nearly independent views of a model. No single view is sufficient.

The UML diagrams provide *multiple perspectives* of the system under analysis or development. The underlying model integrates these perspectives so that a self-consistent system can be analyzed and built. The diagrams, along with supporting documentation, are the primary artifacts that a designer sees, although the UML and supporting tools will provide for a number of views. UML looks for techniques, including component technology, visual programming, patterns and frameworks. UML seeks techniques to manage the complexity of systems as they increase in scope and scale. In particular, the UML recognizes the need to solve iterative architectural problems. One of the key motivations in the minds of the UML developers was to create a set of semantics and notation that adequately addresses all scales of architectural complexity, across all domains. The primary goals of the UML are:

(1) Provide users with a ready-to-use, expressive visual modeling language to develop and exchange meaningful models;
(2) Furnish extensibility and specialization mechanism to extends the core concepts;
(3) Support specifications that are independent of particular programming languages and development processes;
(4) Support higher-level development concepts such a components, collaborations, frameworks and patterns.

With regard to (4) the UML should be tailored as new needs are discovered, and for specific domains, specializing the concepts, notations, and constraints for particular application domains.

### 2.1 Use Case Models

In the UML, one of the key tools for behaviour modeling is the *Use Case* model, originated from the *Object-Oriented Software Engineering* (OOSE) [JCJO92]. Use cases are a way for specifying required usages of a system. Typically, they are used to capture the requirements of a system, that is, what a system is supposed to do. The key concepts associated with the use case model are *actors* and *use cases*. The users, and any other systems that may interact with the system, are represented as actors. Actors always model entities that are outside the system. The required behaviour of the system is specified by one or more use cases, which are defined according to the needs of actors. Each use case specifies some behaviour, possibly including variants, that the system can perform in collaboration with one or more actors. Use cases define the offered behaviour of the system without reference to its internal structure. These behaviours —involving interactions between the actor and the system— may result in changes to the

state of the system and communications with its environment. A use case can include possible variations of its basic behaviour, including exceptional behaviour and error handling. Each use case specifies a unit of useful functionality that the system provides to its users, i.e., a specific way of interacting with the system. The behaviour of a use case can be described by means of *interaction diagrams (sequence and collaboration diagrams)*, *activity charts*, and *states diagrams*, or by pre-conditions and post-conditions, as well as natural language text where appropriate. Which of these techniques to use depends on the nature of the use case behaviour as well as the intended reader.

From a pragmatic point of view, use cases can be used for the specification of the (external) requirements on an entity, and for the specification of the functionality offered by an (already realized) entity. Moreover, the use cases also indirectly states the requirements that the specified entity imposes in its users, i.e., how they should interact so the entity will be able to perform its services. One actor can communicate with several use cases of an entity, i.e., the actor may request several services of the entity, and one use case communicates with one or several actors when providing its service. In the case where subsystems are used to model the system's containment hierarchy, the system can be specified with use cases at all levels, and use cases can be used to specify subsystems and classes. In addition, actors representing potential users describe the particular system views of each user, and inheritance between actors is used for specifying common (inheritance of) view of the developed system.

### 2.2 Generalization and Include Relationships

One of the most controversial elements of the use case model along the UML development has been the Use case relationships named *inclusion*, *generalization* and *extension*, introduced by Jacobson [Jac03]. These relations have an unstable semantics along the UML development. They have received several interpretations [GLQ02, Sim99, MOW03, MOW04], reflecting a high degree of confusion among developers. Our approach for specifying use cases with activity charts is also concerned with the study of use case relationships. In particular, we are interested in the study of the relationships inclusion and generalization. The aim of our work is to provide a more formal definition of use case relationships in terms of their specification by means of activity charts. Our idea is to compare activity charts in order to compare use cases. Such a comparison is abstract and is defined in term of states and transitions included in the activity charts, and interpreted as generalization and <<*include*>> relationships. This topic of research has not been enough explored, although there exist some works which have studied it [Ste01, OP99, Sim99]. On one hand, in [Ste01] use cases are described by means of state machines, and inclusion relationship is studied. On the other hand, in [OP99, Sim99] the cited and extend relationship proposed by Jacobson are discussed but without providing a formal description.

From a pragmatic point of view, an inclusion relationship between two use cases means that the behaviour defined in the target use case is included at one location in the *sequence of behaviour* performed by the base use case. A use case

may be included in several other use cases, and a use case may include several other use cases. In that sense, the included use case represents encapsulated behaviour which may easily be reused in several use cases. A generalization relationship between use cases implies that the child use case contains the sequences of behaviour and participates in all relationships of the parent use case. The child use case may also define new behaviour sequences, as well as additional behaviour, and specialize existing behaviour of the inherited ones. A use case may have several parent use cases, and a use case may be a parent to several other use cases. Therefore, inclusion and generalization relationships are closely related with well-known concepts on object-oriented A&D, such as *encapsulation* and *inheritance*. Up to now, this interpretation has been intended for UML experts, but it needs particular interpretations depending on the diagrams used for specifying use cases.

Due to the iterative perspective of the model-driven development of the UML, the use case view of a system can be refined in the development process, and therefore, inclusion and generalization can also be used for *tracing* the development. System designers can include, generalize and specialize certain use cases which were described in early stages of the development process. In summary, use case model can be used for documenting both the final system and the development process, enabling the maintenance of the system. From the point of view of future system designers, use case model allow the knowledge of the system (components) structure and behaviour and the relationships of integrated components: tasks, requirements, services, external needs, behaviour, and so on. The integration of a new system with the older one should take into account the particular requirements specified on the use case model.

### 2.3 Activity Charts

Activity charts basically describe the set of *states* (and the corresponding *transitions*) which a given entity follows when a given service is required. Therefore, activity charts can be used for specifying the *internal behaviour* of a certain entity or subsystem of the developed software, such as a database, a graphical interface, a software component, or any specific software. Depending on the nature of the entity to be specified, the states can describe *internal states*, that is, states in which no interaction with the environment is achieved, and *external states* which involves communication with actors and other entities. In addition, transitions can be also subdivided following the same criteria, that is, *internal transitions* achieved by the entity and *external transitions* with participation of the outside world.

In our approach, an activity diagram is used for specifying the set of behaviour sequences of a given use case, allowing the detection of included use cases (included sequences of behaviour) as well as more general and particular use cases (generalizing and specializing sequences of behaviour). With this aim, we need to define an abstract definition of similar properties on activity diagrams, named inclusion and generalization. Given that activity diagrams handle states and transitions, which describes behaviour sequences, the inclusion and general-

ization relationships between activity diagrams are defined in terms of behaviour sequences. The case of inclusion is simpler describing *subsequences of behaviour*, but generalization uses in its definition an abstract *replacement relationship*. It can be identified with the usual concept of replacement of object-oriented A&D, that is, a certain component offers the same functionality (eventually, adding new functionality) as another one, and therefore the second one can be replaced by the first. This generic view of replacement induces a generic generalization relationship on activity diagrams, which induces the same use case relationship. Therefore use case modeling offers an object-oriented system view.

## 3 A Method to Describe Use Cases with Activity Charts

In this section, we will show how to describe use cases with activity charts. With this aim, we present a method based on the identification of use case relationships from the description of activity diagrams.

### 3.1 Identifying Actors and Use Cases

Firstly, the developer describes the functional requirements of the system by means of a use case diagram. A use case diagram consists of a set of actors (users and external systems) and use cases.

Relationships between actors are *generalizations*. An actor $p$ is more general than another actor $q$ whenever $q$ can interact with the system as $p$ and additionally, can interact in more cases. The developer should also identify the set of use cases related to each actor that will represent the set of tasks to be achieved by the actor. Relationships between actors and use cases are called *associations*. In our method, generalizations and associations may be identified in the first step, but they can be refined later. This first developing process will get a (still non formal) use case diagram.

### 3.2 Describing Use Cases with Activity Charts

Once the actors and use cases associated with each actor have been specified, the developer should provide in a second step a set of activity charts to describe each use case in the use case diagram. They may be basically specified in the early stages of the development process and refined later.

Activity charts are graphs linking *states* by means of *transitions*, which are arrows connecting an *origin state* and an *end state*. There are two special states: *initial* and *final* states. Initial (resp. final) state is the starting (resp. end) point of the activity chart. Each state may represent a system state and transitions can represent actor interactions (user events, external system calls) or internal system behaviour (execution steps and threads).

Transitions are labeled by means of *conditions/actions*, representing conditions to be hold and actions to be achieved for state change. There can be *diamonds* between transitions describing alternative paths depending on a *boolean*

*condition*. An state can also be described by means of a separate activity chart, describing (sub)states and transitions performed in the state. In this case, the state is called *non-terminal state*, and otherwise it is called *terminal states*.

### 3.3 Identifying Activity Chart and Use Case Relationships

The third (and last) step consists on the identification of use case relationships from the described activity charts. The relationships between uses cases are <<*include*>> dependencies, together with generalizations. Here, the developer should apply the following rules to compare activity charts, which induces the <<*include*>> and generalization relationships between use cases. This provides a new more formal and refined use case diagram in which there have been specified the cited use case relationships. Both activity diagrams and use case diagram can be refined in later stages of the development process when the knowledge of the system requirements is refined. That is, use case relationships can be late detected due to the refinement of the activity diagrams.

**Identifying Activity Chart Relationships.** Firstly, we have to assume that terminal states and transitions of activity charts can be compared by means of a (reflexive) *replacement relationship* $\sqsubseteq$ in such a way that two (terminal) states satisfies $s \sqsubseteq s'$ (or two transitions $\lambda \sqsubseteq \lambda'$) whenever $s'$ can be replaced by $s$ (or $\lambda'$ can be replaced by $\lambda$). The replacement relationship can express similar semantics (or behaviour). The replacement relation should be decided by the developer.

This replacement relationship induces a *replacement relationship on activity charts* in such a way that an activity diagram $a'$ can be replaced by $a$ if the states and transitions of $a'$ can be replaced by the states and transitions of $a$.

In addition, activity charts can be compared by means of an *inclusion relationship*. Inclusion can be intended as an activity chart that includes another one, but without changing the behaviour, that is, states and transitions are not modified, and neither new states or transitions can be added over the included activity chart. In practice, if an activity chart $a$ includes an activity chart $a'$ then one of the states of $a$ is $a'$, although $a$ may implicitly include all the states and transitions of $a'$. For simplicity, we assume the first case.

Obviously, some activity charts may not be compared by means of replacement and inclusion relations, this means they do not describe "similar" activities. However, some activity diagrams can be compared by means of a combination of replacement and inclusion and not by a single one. In this case, in order to be compared, there should be defined *intermediate* activity charts (decomposing the activity charts), in such a way the original activity diagrams could be compared through *a chain of relationships*. In the whole development process, intermediate activity diagrams can be detected when specifying new users interactions of refinement of the existent ones.

**Identifying Use Cases Relationships.** Once we can compare activity charts by means of the above relationships, the use cases —which are described by means of activity charts— can also be compared by means of the following relationships.

— A use case $u$ *includes* a use case $v$ whenever the activity chart of $u$ includes the activity chart of $v$.
— A use case $u$ is more general (*generalizes*) than other $v$ whenever there exists an activity diagram $u'$ such that $u$ can be replaced by $u'$ and $v$ includes $u'$.

In the UML there are some additional information about the use case diagrams like roles, multiplicity, directionality, and *extend* dependencies, but they will not be considered in our approach yet.

## 4 A Case Study for GUI Design

In order to clarify the cited concepts, in this paper we present a case study that puts into practice our approach. This case study handles the use case model for designing *Graphical User Interfaces* (GUI). Our general method can be specialized depending on the nature of the system to be developed or the part of the system to be build. Although we have decided to apply our method for designing GUIs (that is, to describe the presentation logic of the system), however, it does not means that the first view of the system to be developed is the user interface. Following the UML philosophy, many views of a system can be built in the early stages and refined in later steps, such as business and data logic views. Each stage complements each other, providing a multiple view perspective of the developed system. We have chosen the user interface view as case study given that by its simplicity the main concepts of our approach (i.e., inclusion and generalization) can be detected in user interface design.

Graphical user interfaces have become increasingly dominant, and the design of the "external" or visible system has assumed increasing importance. This has resulted in more attention, being devoted to usability aspects of interactive systems, and a necessity of tool development that supports the design of the presentation logic. Models and notations are required to describe user tasks, and map these tasks on to the user interface. The user interface (as a significant part of the most applications) should also be modeled using UML. However, it is by no means always clear how to model user interfaces using UML, although there are some recent approaches [Kov98, dSP03, dSP00, EK00, EKK99, Nun03, BNT02] which have addressed this problem. The proposals of [dSP03, dSP00, Nun03] identify some aspects of GUI that cannot be modeled using UML notation, and a set of UML constructors that may be used to model GUI. However, a methodology for GUI design using the use case model is not completely addressed, and there also exists a lack of formal description of use cases and a correspondence between use case relationships and GUI components.

Another similar work to our contribution is [EK00, EKK99] in which state machines and Petri-nets are used to specify GUI in UML. In the quoted approaches they specify user interaction but they also lack of use case relationships handling.

In our case study, activity charts are used for describing user interaction. In particular, they describe the presentation logic of an *applet-based system* (similar to [EK00, EKK99]), in which a set of applet windows are shown to the user, and the user interacts with them in order to put and get data from the system. With respect to this choice, our aim is not to constraint the implementation to a particular window system but the acceptance of the *JAVA swing classes* between developers allows us to assume the reader is familiarized with the implementation details of the GUIs. Our framework can be adapted to others user-interface development technologies with a bit of effort, adding new UML stereotypes for both input and output components, and making similar mappings between use cases and other kind user windows.

### 4.1 Specializing Activity Charts

In the activity charts, states represent outputs to the user which are labeled with UML *stereotypes* representing visual components for data output. Transitions represent user inputs which are labeled with UML stereotypes representing visual components for data input and choices. Once activity diagrams describe each user interface for each actor in the use case model, the inclusion relationship describes subsequences of interaction with the graphical interface and the generalization relationship inheritance of common (eventually, more particular) tasks and interactions between several users.

The activity charts can be specialized for describing user interaction as follows:

— Each state of the activity chart describing a use case necessarily falls in one of the two following categories:
  - A terminal state is labeled with a *UML stereotype* representing an *output GUI component*. Therefore, they are also called *stereotyped states*.
  - A non-terminal state is not labeled and is described by means of an activity chart. The non-terminal states can be "use cases" of the use case diagram or not.

— Each *transition* in the activity chart of a use case can be labeled by means of *conditions* or *UML stereotypes with conditions*. The UML stereotypes represent *input GUI components*. This kind of transitions is also called *stereotyped transitions*. The conditions represent *use choices or business logic*.

With respect to the relation of replacement, it is useful for instance to reuse some design artifacts of a model (i.e., a collection of states, transitions, stereotypes), or change the functionality of an artifact. In the case study the stereotyped states can be replaced whether the *output GUI component* can be replaced. For instance, a list with two columns can be replaced by another list with three columns without lost of functionality. The same happens with stereotyped interactions which can be replaced if the *input GUI components* can be replaced.

For instance, a selection of any of the cited list. Finally, conditions can be, for instance, replaced if one of them is *more restrictive* than the other.

### 4.2 Building GUIs

Now, we can build our GUI from the use case diagram and the set of activity charts following the next rules:

— Each *actor* representing a user in the use case diagram is an *applet*[1]. Actors representing external systems are not considered for visual component design.
— The *generalization relationships* between two actors $p$ and $q$ ($p$ generalizes $q$) corresponds with *inheritance* of the applet represented by $q$ from the applet representing $p$[2].
— Each *use case* in the use case diagram is an *applet*[3].
— The *generalization relationship* between two use cases $u$ and $w$ ($u$ generalizes $w$) corresponds with *inheritance* of the applet representing $w$ from the applet representing $u$.
— The <<*include*>> *relationship* between two use cases $u$ and $w$ ($u$ includes $w$) corresponds with the *invocation* from the applet representing $u$ of the (sub)applet representing $w$[4].
— In the non-terminal states case, the use case diagram can specify <<*include*>> or generalization relationship between the non-terminal state and the use case, and we follow these rules:
  - In the <<*include*>> relationship case, the non-terminal state is also an applet and contains the GUI components in the associated activity diagram.
  - In the generalization relationship case, the non-terminal state is also an applet containing the GUI components in the associated activity diagram, but the use case also contains these GUI components.
— The non-terminal states —which does not appear in the use case diagram— are not applets, and the GUI components in the associated activity diagrams are GUI components of the applet of the use case.
— The conditions of the transitions of an activity chart are not taken into account for the GUI design.

[1] For some complex user interfaces, the applet class could be replaced with other specialized swing classes like the frame class, which enables to build a single window in which the complete functionality of each actor is presented.
[2] When using frames we can use inheritance of frames.
[3] When using a frame for implementing an actor, a use case connected with an actor can be implemented either with a separate applet or using a window area, and an applet built in the main frame.
[4] When using frames, the invocation can be replaced by the built-in of the applet.

## 5 The Internet Book Shopping (IBS) Example

In this section, we will explain a simple example of an Internet book shopping (IBS) that illustrates the functionality of our proposed method. In the IBS there will basically appear three actors: (a) the customer, (b) the ordering manager, and (c) the administrator. A customer actor directly carries out the purchases by the Internet. The customer can also consult certain issues of the product in a catalogue of books before carrying out the purchase. On the other hand, the manager deals with (total or partially) customer's orders. Finally, the system's administrator actor can manage the catalogue of books adding and removing new books in the catalogue or modifying those already existing. The administrator can also update or cancel certain component characteristics of an order or those orders fulfilling certain searching criteria. Furthermore, both the manager and the administrator should identify themselves before carrying out any kind of operation restricted to his/her environment of work.

Considering this framework, in the next sections we will describe an IBS GUI project that illustrates the proposed method, i.e., how to connect the system's use cases description to activity charts. This connection helps the developer to translate and/or connect some analysis features (i.e., use cases descriptions) to/with behavioural descriptions of the system. In our case, these behavioural descriptions only will concern with activity charts.

### 5.1 Modeling the IBS System Using Use Cases

Let's suppose that the developer of the IBS system wishes to model the *presentation logic* by means of use cases. Initially, the use case diagram contains the identified actors of the system: in our IBS example, the actors are the `Customer`, the `Manager` and the `Administrator`.

To model the presentation logic of the IBS system, the developer should previously identify all those future windows of the system (graphical user interfaces), carrying out some quick outline of their content[5]. The connection of an actor with one or more use cases in the use case diagram will be interpreted as a set of options (menu) on a first window on which the actor will interact with the system. Figure 1 shows some prototypes of those windows of the IBS presentation logic that the developer wants to reach[6].

Initially, the use case diagram, which models the IBS presentation logic, contains three kinds of interfaces identifying the actors of the system. A first non-formal description of the IBS system using a use case diagram is shown in

[5] As we said before, the model-driven development follows an iterative refinement process and therefore, here, we are supposing the designer has a well-defined collection of the GUI requirements.

[6] The windows shown in Figure 1 are only a pragmatic simple example as result obtained after applying our method. We suppose that the designer of the system has decided (for any reason) this window organization. We do not consider whether the distribution is suitable or rationale.

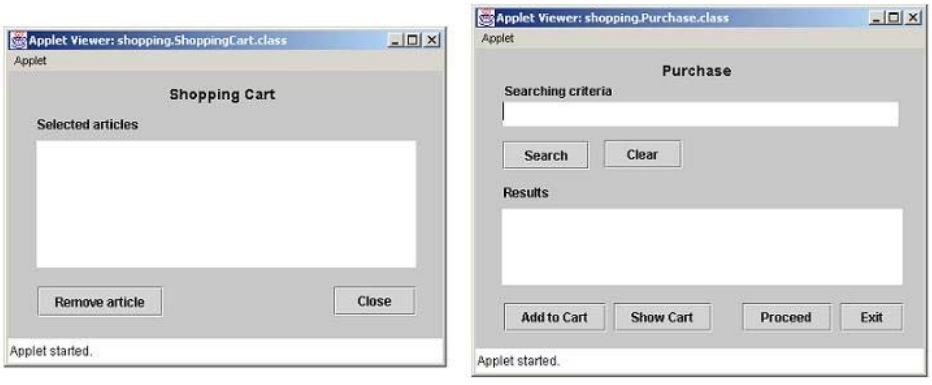



**Fig. 1.** Some windows of the IBS presentation logic

Figure 2. The diagram also contains all the high-level functions represented by means of use cases. For example, note that the administrator's interface agrees to the *presentation logic* through three possible options (use cases): `Manage catalogue`, `Update orders` and `Update partial orders`. Something similar happens to the remainder actors of the system.

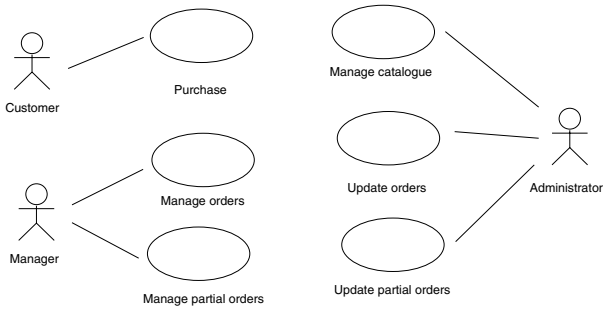



**Fig. 2.** A non-formal use case description of the IBS example

As we have mentioned in the previous section, the non-formal definition of the system will be refined, causing more precise use cases diagram(s). Figure 3 shows a more complete *presentation logic* definition for the IBS system. In this case, we have chosen to include all the *presentation logic* in a single use case diagram, although it could be itemized in more than one to deal with more structured use case descriptions.

Let's see now certain aspects of the use case diagram shown in the figure. Firstly, we emphasize the use given to the relationships *<<include>>* and *generalizations*. In our method, the *<<include>>* relationship can be used to represent optional or mandatory behaviour. This two kinds of relationships are properly modeled and interpreted in the activity charts associated with both con-

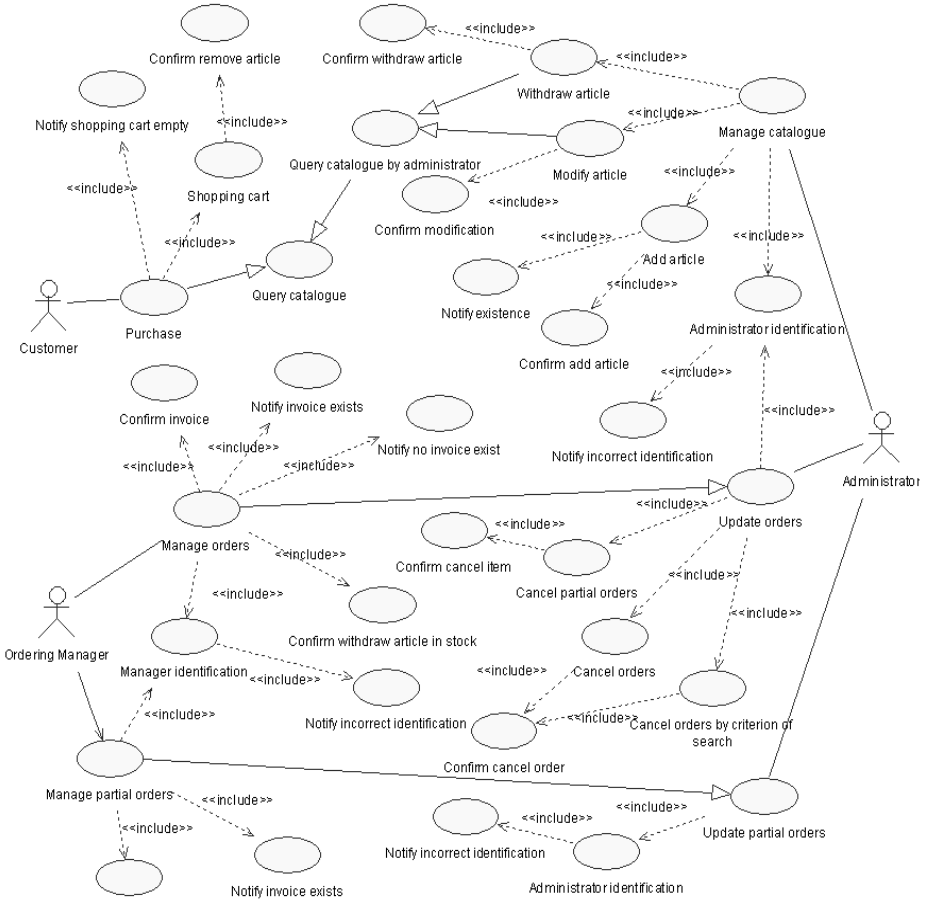



**Fig. 3.** The refined IBS use case diagram

nected use cases, since use case diagram does not distinguish between hard/soft dependencies (i.e., mandatory/optional relationships). For instance, the use case `Manage catalogue` is an applet that directly depends on four use cases, connected to them by means of a <<*include*>> relation. However, these connections are similar to three of them, and different from the others. The <<*include*>> relationships between the use cases `Withdraw article`, `Modify article` and `Add article` were modeled by the system's designer as relations of optionally (the branches of the use case `Manage Catalogue`'s behaviour go to these states in the activity chart). However, the use case `Administrator identification` was considered by the system's designer as a relation of mandatory (this state is always reached in the activity chart) of the use case `Manage Catalogue`.

Therefore, an <<*include*>> relationship can mean that a use case could be considered as a composition of two or more other use cases. For instance, the use case `Manage catalogue` is composed of the use cases `Withdraw article`,

`Modify article` and `Add article` (i.e., applets or frames). An *<<include>>* relationship can also indicate that a use case mandatory depends on another use case to operate. For example, since the administrator should be itself identified before working with the system, the three use cases which he/she directly operates with (i.e., `Manage catalogue`, `Update orders` and `Update partial order`) mandatory depend on the `Administrator identification` use case.

In our case, the relation of *generalization* is intended as an inheritance of behaviour (i.e., GUI components). For example, the use case `Query catalogue` has been established as a generalization of the use case `Purchase`. This relation will mean that, due to own reasons, the developer of the system wishes to model the purchasing process re-using and modifying the functionality of the query process. Note how the use case `Query catalogue by administrator` also inherits from query catalogue and generalizes the use cases `Withdraw article`, and `Modify article` connecting them to a part of the client's *presentation logic* and the administrator side

The distinction between *include* relationships (mandatory, optional) and generalization is established by the system's designer into the activity charts of those include-connected use cases. In the following sections we will only focus on the `Purchase` use case to explain the behaviour of the method.

### 5.2 Mapping the IBS Features, Use Cases into Activities

As the developer stated, each use case modeling the presentation logic will correspond with an applet (or frame) component. Activity charts describe certain graphical and behavioural details about the graphical components of an applet. In our case study, we have only adopted four JAVA graphical components: `JTextArea`, `JList`, `JLabel` and `JButton`. Nevertheless, other graphical elements could be easily considered in the activity chart since they are modeled as state or transition stereotypes.

Graphical components can be classified as input (a text area or a button) and output components (a label or list). Input and output graphical components are associated with terminal states and transitions by using the appropriate stereotype, for instance, `JTextArea`, `JList`, `JLabel` stereotypes are associated with states and `JButton` stereotype to transitions. Since the graphical behaviour concerns to states and transitions, next we will describe them separately.

A **state** can be stereotyped or not. Stereotyped states represent terminal states which can be labeled by the `<<JTextArea>>`, `<<JList>>` and `<<JLabel>>` stereotypes. For instance, Figure 4 shows the activity chart for the `Purchase` use case. This diagram shows the graphical and behavioural content of the applet window where the purchases can be carried out. The activity chart is composed of four states. Two of them are terminal states, since they correspond to graphical elements. They are stereotyped (`<<JTextArea>>`) and labeled by a text related to the graphical element. Two other states have been described in a separate activity chart in order to structure better the design. The name of a separate activity chart should be the same as the one of the state.

The activity chart's behaviour, in Figure 4, shows how the customer begins the purchasing process of querying, adding or removing articles of the shopping cart. After a usual purchasing process, the shopping system requests the customer a card number and a postal address to carry out the shipment, whenever the shopping cart is not empty.

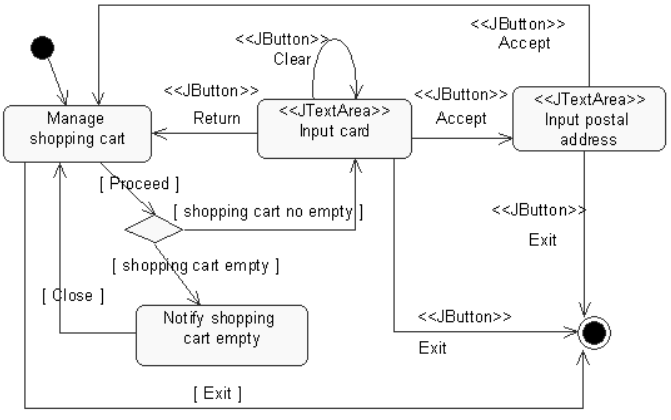



**Fig. 4.** The activity chart for the `Purchase` use case

According to the GUI developer's rules, if a state is not labeled with a stereotype, this means that the state is described in another activity chart. This new activity chart can either represent the behaviour of another use case or simply a way of allowing a hierarchical decomposition of the original activity chart. For example, in the activity chart associated with the `Purchase` use case (see Figure 4), there appear two non-terminal states: `Manage shopping cart` and `Notify shopping cart empty`. At the same time, two activity charts are described for both states. All these activity charts are shown in Figure 5.

In the activity chart of the `Notify shopping cart empty` use case, we can observe how the target use case (being modeled) brings to an activity chart. The model represents a warning applet window containing only the text `Shopping cart empty` and the button `Close` to close the warning window. In the `Manage shopping cart` activity chart, the states `Query catalogue` and `Shopping cart` are itemized on independent activity charts. Both states would also correspond with an applet, since they appear as use cases in the use case diagram.

**Transitions** in the activity chart can be labeled by means of *stereotypes*, *conditions* or both together. For instance, a button is connected to a transition by using the `<<JButton>>` stereotype, and the name of the label is the name of the button. For example, a `Show cart` transition stereotyped as `<<JButton>>` will correspond with a button component called "Show cart".

**Conditions** can represent *user choices* or *business/data logic*. The first one is a condition of the user's interaction with a graphical component (related to button or list states), and the second one is an internal checking condition (not

related to the states, but to the internal process). For example, in our case study the selections on a list are modeled by conditions. Note in the Query Catalogue activity chart shown in Figure 5 (b), the list `Results` is modeled by a `<<JList>>` state and the condition `[Selected article]`. Figures 4 and 5 show transitions (p.e., `[Close]`, `[Exit]` or `[Proceed]`) that correspond with conditions of the kind *user choice*. The `[Exit]` output transition of the state `Manage shopping cart` means that the user has pressed a button called `Exit`, which has been defined in a separate `Manage shopping cart` activity chart (see Figure 5). Nevertheless, the `[shopping cart no empty]` and `[shopping cart empty]` conditions are two *business/data logic* conditions, in which the human factor does not participate.

Furthermore, stereotyped transitions (buttons in our example) and conditions connect (non) terminal states to (non) terminal states. As we said before, a condition would be an output of a non-terminal state in case the user interacts with a button or a list component inside the respective non-terminal state. The usual way "condition/action" transition can connect (non) terminal states to (non) terminal states. A condition/action transition between states means which condition should be present to achieve the action. In our case study, an action can only be a button. For instance, to remove an article from the shopping cart, it must previously be selected from the cart list (Figure 5, c).

Condition/action transitions are also useful to model the behaviour of the generalization relationships between use cases in a use case diagram. Note in the original use case diagram how the `Purchase` use case inherits the behaviour of the use case `Query catalogue` by means of a generalization relationship.

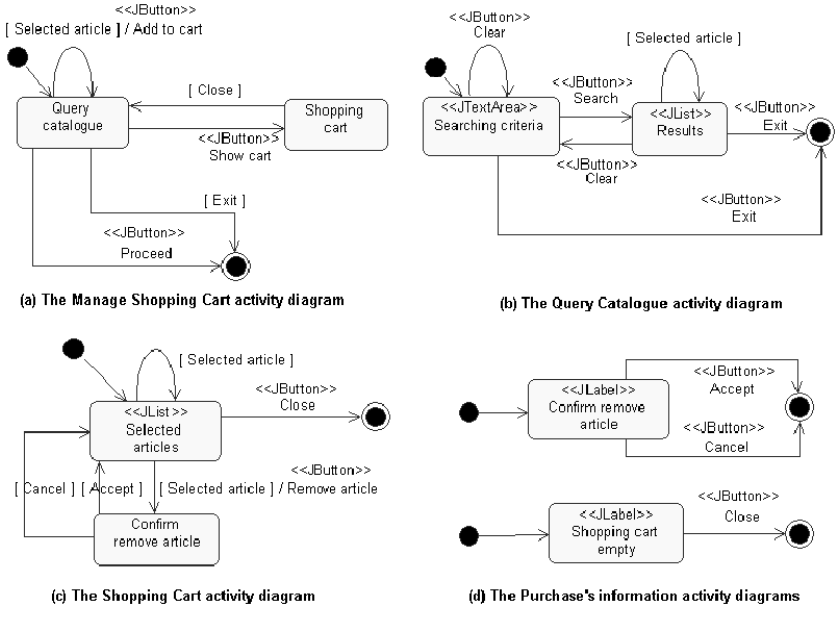



**Fig. 5.** The whole activity chart of the `Purchase` use case

This inheritance behaviour is modeled in the Purchase activity chart as a non-terminal state that includes the behaviour of the Query Catalogue activity chart. For example, let us observe the behaviour of the query catalogue shown in Figure 5 (b). In this activity chart, the user introduces the searching criteria in the text area, presses the button `Search` and then the results are shown on a list. After that, the user can select articles in the list, presses a button to exit or try a new search by pressing the button `Clear`. Thus, when the Purchase use case inherits the Query Catalogue use case, it should be possible to interrupt its behaviour.

Condition/action transitions can be used to interrupt an inherited behaviour. For example, the query catalogue's behaviour (previously described) is adopted in the activity chart of the Purchase use case as a non-terminal state called `Query catalogue` (see Figure 5, a). The output transition `[Selected article]/Add to cart` mean that the `Add to cart` button at the Purchase applet (use case) can interrupt the query catalogue behaviour whether an article has been selected (condition). Analogously, the output transitions `Proceed` and `Show cart` at the Purchase applet (use case) mean that both the `Proceed` and `Show cart` buttons can interrupt the inherited behaviour of the query catalogue.

On the other hand, a generalization relationship does not only represent an inheritance of the behaviour as an extension; for instance, the Purchase use case inherits the Query Catalogue use case and increases its behaviour to hold the buttons `Add to cart`, `Show cart` and `Proceed`. However, a generalization relationship can also deal with a **replacement** of behaviour instead of an increase in behaviour. For example, note in the original use case diagram how the Query Catalogue by Administrator also inherits the Query Catalogue. Let us suppose that their behaviours (activity charts) are the same, but the results list shown to the customer actor (the `Results` state) is different from that shown to the administrator actor (for instance, `Administrator Results` state). In this case, the system's designer can use the behaviour (activity chart) of the Query Catalogue use case to model the behaviour (activity chart) of the "Query Catalogue by Administrator" re-writing (replacing) the results list (p.e., replacing `Results` by `Administrator Results`). This rule of replacement can also be considered on transitions (p.e, replacing a button by another GUI component). Finally, the conditions and "conditions/actions" can be also replaced. In all cases, is a decision of the designer to allow the replacement of states and transitions.

To develop the IBS project example we have used the Rational Rose for Java tool. For space reasons, we have included here just a part of the GUI project developed for the case study. A complete version of the project is available at `http://www.ual.es/~liribarn/Investigacion/usecases.html`.

## 6 Conclusions and Future Work

In this paper, we have studied a method for describing use cases by means of activity charts based on a Model-Driven Development (MDD) perspective [OMG03]. The use case diagrams help the developer to identify the requirements of the system and to study its high level functionality. The activity charts

allow us to discover new behavioural details of the system or to describe better the already existing. Nevertheless, in this paper we have shown how a direct correspondence between the requirements identified in the use cases with these UML activity diagrams is feasible. The technique also allow us to identify the two main use case relationships (include and generalization) by means also of activity charts. Through a case study, we have shown how our technique can be applied in GUI-oriented component development for rapid prototyping of the external view of the system. Although in our approach we can completely specify the behaviour of the GUI components of the system, in the early stages of the system development a basic behaviour can be specified which could be refined in later stages. We have chosen the JAVA swing classes and components for describing user interfaces due to the acceptance of this technology (*applet*, *frames*, and *events*), however our approach can be adapted to other technologies for GUI building (for instance, *hypertext* and *hyperlinks*). Finally, the use case model together the GUI specification with activity diagrams provide a description of the behaviour and structure of the GUIs of the developed system, and in general can be used for describing a library of reusable GUI components.

As a future work, we firstly plan to apply our general method to other parts of the system (i.e. business or data logic). Secondly, we would like to extend our work to deal with the <<*extends*>> relationship of use cases. Thirdly, we would like to formalize and to incorporate our method in a CASE tool in order to automate it. And finally, we would like to integrate our technique in the whole development process.

## Acknowledgements

The authors would like to thank the anonymous referees for their insightful comments, that greatly helped them improve the contents and readability of the paper. We would also like to thank to the attendees of MIS'04 for the suggestions about our paper. This work has been partially supported by the Spanish project of the Ministry of Science and Technology "INDALOG" TIC2002-03968 under FEDER funds.

## References

[BNT02] R. Biddle, J. Noble, and E. Tempero. Essential use cases and responsibility in object-oriented development. In *Australasian Computer Science Conference (ACSC2002)*, 2002.

[Boe88] B. W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, 21(5):61–72, May 1988.

[dSP00] P. P. da Silva and N. W. Paton. User interface modelling with UML. In *Information Modelling and Knowledge Bases XII*, pages 203–217. IOS Press, 2000.

[dSP03] P. P. da Silva and N. W. Paton. User Interface Modeling in UMLi. *IEEE Software*, 20(4):62–69, 2003.

[EK00] M. Elkoutbi and R. K. Keller. User Interface Prototyping Based on UML Scenarios and High-Level Petri Nets. In *International Conference on Application and Theory of Petri Nets (ICATPN 2000)*, pages 166–186. LNCS 1825, 2000.

[EKK99] M. Elkoutbi, I. Khriss, and R. K. Keller. Generating user interface prototypes from scenarios. In *IEEE International Symposium on Requirements Engineering (RE '99)*, page 150. IEEE Computer Society, 1999.

[GLQ02] G. Génova, J. Llorens, and V. Quintana. Digging into use case relationships. In *International Conference on the Unified Modeling Language (UML 2002)*, pages 115–127. LNCS 2460, 2002.

[Jac03] I. Jacobson. Use Cases – Yesterday, Today, and Tomorrow. Technical report, Rational Software, 2003.

[JCJO92] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering: a Use Case Driven Approach*. Addison-Wesley, 1992.

[Kov98] S. Kovacevic. UML and User Interface Modeling. In *International Conference on Unified Modeling Language (UML'98): Beyond the Notation*, pages 253–266. LNCS 1618, 1998.

[MOW03] P. Metz, J. O'Brien, and W. Weber. Specifying use case interaction: Types of alternative courses. *Journal of Object Technology*, 2(2), March-April 2003.

[MOW04] P. Metz, J. O'Brien, and W. Weber. Specifying use case interaction: Clarifying extension points and rejoin points. *Journal of Object Technology*, 3(5), May-June 2004.

[Nun03] N. J. Nunes. Representing User-Interface Patterns in UML. In *International Conference on Object-Oriented Information Systems (OOIS 2003)*, pages 142–151. LNCS 2817, 2003.

[Nus01] B. Nuseibeh. Weaving together requirements and architectures. *IEEE Computer*, 34(3):115–117, March 2001.

[OMG03] OMG. Unified Modeling Language Specification, version 2.0 (MDA). Technical report, Object Management Group, June 2003.

[OP99] G. Övergaard and K. Palmkvist. A Formal Approach to Use Cases and Their Relationships. In *International Conference on the Unified Modeling Language (UML'98): Beyond the Notation*, pages 406–418. LNCS 1618, 1999.

[Sim99] A. J. H. Simons. Use cases considered harmful. In *International Conference on Technology of Object-Oriented Programming Languages and Systems (TOOLS-29 Europe)*, pages 194–203. IEEE Computer Society, 1999.

[Ste01] P. Stevens. On Use Cases and Their Relationships in the Unified Modelling Language. In *International Conference on Fundamental Approaches to Software Engineering (FASE'01)*, pages 140–155. LNCS 2029, 2001.

# Spatial Constraint Modelling with a GIS Extension of UML and OCL: Application to Agricultural Information Systems

François Pinet[1], Myoung-Ah Kang[2], and Frédéric Vigier[1]

[1] Cemagref, Clermont Ferrand, France
{francois.pinet, frederic.vigier}@cemagref.fr
[2] Laboratory of Computer Science, Modelling and System Optimisation (LIMOS), ISIMA / Clermont Ferrand University, France
kang@isima.fr

**Abstract.** In numerous cases, the modelling of Geographical Information Systems (GIS) is a difficult task. This fact is partially due to the natural complexity of spatial types and invariants. Based on this observation, the present paper aims at specializing the Unified Modelling Language (UML) and its associated Object Constraint Language (OCL) in order to facilitate the design of GIS. A complete study of the proposed extensions will be presented as well as several in-depth experiments in the domain of agricultural information systems.

## 1 Introduction

This paper focuses on the description of a specific formalism for spatial data modelling and a complementary language for the expression of topological constraints. The proposition aims at enabling designers to easily and clearly specify aspects related to spatial information. For that, our proposition uses concepts that are easy to understand for Geographic Information System (GIS) designers. Indeed, the main goal of our works is to respond to the needs of information system designers in offering methods more adapted to the field of GIS than classical modelling languages. Our contribution allows to describe the complexity of both geographic types and spatial constraints.

Firstly, this paper tries to formalize the integration of the concept of geographical type constraints in the Unified Modelling Language (UML) [11]. The purpose of this extension is to specialize UML in order to clarify and facilitate the specification of spatial data types. In order to reach this goal, we use the concept of geographic class i.e. a class that has an attribute used to store geometries. This paper underlines that type constraints applied on class diagram are important to describe more precisely spatial features. These constraints are related to the spatial attribute of a geographic class in order to define more precisely what types of geometries are associated to the class (a point, a polygon, a point or a polygon, a set of polygons…). The constraints must be able to express the spatial aggregation (or composition), since it is an essential operation in GIS.

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 160 – 178, 2004.

Secondly, beyond the needs of modelling geographic types, the main part of the present paper responds to designer requirements concerning the expression of topological constraints in class diagrams. In order to model these constraints, the existing propositions suggest to draw relationships between classes [6], [12] (see the example in figure 1). This type of representation cannot express constraints depending on a specific condition (for example IF … THEN the constraint is applied ELSE …). Thus, in order to define topological constraints more precisely, an extension of OCL (Object Constraint Language)[1] will be proposed and experimented; this extension consists in adapting existing OCL constructs to geographical needs. The major advantage of this approach is to benefit from OCL functions in order to express topological constraints. The final goal of this extension is to give the capability to generate automatically inside a GIS, a mechanism for integrity checking from a conceptual modelling of spatial constraints.

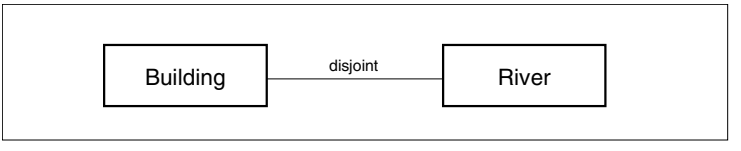



**Fig. 1.** Two classes linked by a relationship "disjoint" in a class diagram; buildings and rivers are geographically disjoint

We experiment our proposition for the design of agricultural information systems (and more precisely in the context of sludge spreading management). The design of this information system type is currently under way at Cemagref and a goal of our OCL and UML extensions is to support this development. Thus, throughout this paper, we will illustrate our proposition in the field of agricultural information systems.

Section 2 of this paper introduces the UML extension dedicated to geographic type modelling. Section 3 deals with the OCL extension for topological constraint design. Last section concludes and draws some perspectives for future work notably concerning the automatic triggers generation.

## 2 Modelling Geographical Type Constraints

Several studies show that the classical modelling languages like UML are not suitable for the design of GIS [1], [5], [6], [7], [8], [12]. The difficulties are related to the expression in class diagrams of the geographic types complexity. The question is, how to represent geometries of objects with UML (e.g. geometries of `Building` objects dedicated the be handled or to be displayed on a map)? The classical

[1] OCL is the new generation constraint language of UML. This language has been developed by IBM and currently, the standard is maintained by the Object Management Group. A growing number of information system designers use OCL in complement of UML models [15].

modelling consists in defining additional classes like `Point`, `Polyline` and `Polygon`. Then, these classes are linked by associations to the other elements of the diagram; unfortunately, with this method, the drawing is rapidly "overloaded" by associations. The needs related to an "easy to read" representation of spatial data features implied the propositions of several UML extensions [1], [5], [8]. These different works implicitly use the concept of geographical type constraints i.e. constraints that are related to the spatial attribute of a geographic class in order to define what types of geometries are associated to the class.

Existing propositions are informal and their presentations are usually made "by example". Thus, the contribution of this section is to suggest a precise formalization of geographical type constraints. Our proposition is based on operations presented in [1]; these operations are really interesting in the GIS domain because they can express spatial aggregations.

In order to know in what cases two constraints are equivalent, the semantics of type constraints will be defined (see appendix A). This theoretical foundation can have several applications concerning the implementation of an algorithm checking if two constraints have exactly the same semantics i.e. if two classes of objects can store the same type of geometry.

### 2.1 Geographic Class and Type Constraint

The concept of geographic class is introduced in order to clearly identify the geometry associated to an object. More precisely, each geographic class instance has a geographic feature. In practice, a geographic class includes an attribute having a spatial type i.e. the value of this special attribute can store a geometry. In this paper, this attribute will be called `geometry`.

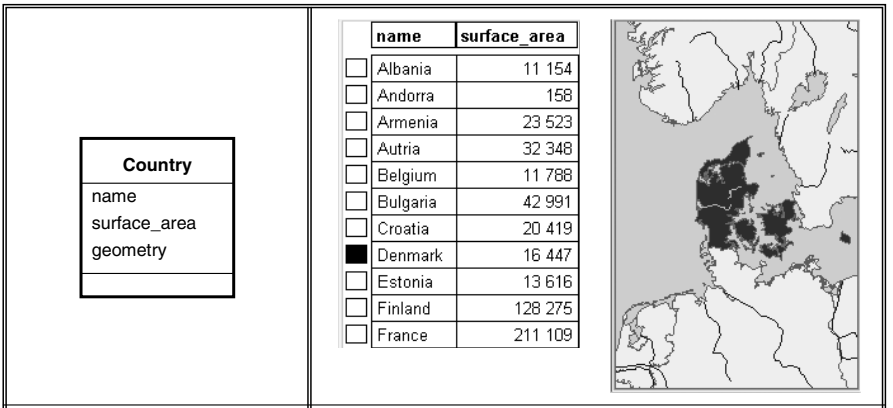
For example, a `Country` class can be viewed as a geographic class because each object of this class is associated to the geometry of a country. In order to describe real world entities, the `geometry` attribute can store data that are issued from spatial aggregation; for example, a geometry of a `Country` class object can be composed by several simple polygons (continental parts and islands).

This representation based on the use of a `geometry` attribute is especially suitable for spatial data design because it corresponds to data structures often used in geographic databases; for example, MapInfo supports this format [9] (see figure 2). The implementation of a geographic class in a GIS is usually called a geographic layer.

Constraints can be associated to geographic classes in order to indicate the types of geometry found in attribute values. Thanks to the constraint specification, in the example of the `Country` class, it becomes possible to model that each value of the `geometry` attribute corresponds to several simple polygons. As presented below, these constraints (called geographical type constraints) are defined by textual expressions in which three types of operators can be combined:

– `A XOR B`; exclusive disjunction operator between `A` and `B`. For example "`polygon XOR point`" indicates that the value of the `geometry` attribute only contains either one polygon either one point (but not both).

- `A AND B`; conjunction operator between `A` and `B`. "`polygon AND polyline`" indicates that the value of the `geometry` attribute is only composed of one polygon and one polyline.
- `A MULT m`; multiplicity operator implying `A` and a multiplicity *m*. "`polygon MULT (1..5)`" indicates that the value of the `geometry` attribute is only composed of "from 1 to 5 polygons".




**a. Country Class defined in database specifications**

**b. Instances of the Country Class** (alphanumeric features are presented in a table and values of the geometry attribute appear on a map).

**Fig. 2.** Implementation of a `Country` class in a MapInfo database and example of instances. The features of a country are composed of three attributes (`name`, `surface_area` and `geometry`). The `geometry` attribute has a cartographic representation and each of its values can have a complex geographic type (such as a composition of simple polygons corresponding to different parts of a country)

Thus, the type constraint "`polygon XOR (point AND polygon)`" associated to a class `C` indicates that the `geometry` attribute of `C` must only contain: a) one element having the `polygon` type, or b) a composition of one element having the `point` type with one element having the `polygon` type. In other words, the value of the `geometry` attribute must contain only "one polygon" or only "one point and one polygon". The previous constraint is completely equivalent to "`polygon AND (point MULT (0..1))`".

Also, the type constraint "`(point AND polygon) MULT (1..*)`" expresses that the `geometry` attribute value must only contain point(s) and polygon(s); according to the constraint, the `geometry` attribute value must contain the same number of points as the number of polygons.

Note that a same type of operators can be found several times in a constraint expression e.g. "`point XOR polyline XOR polygon`". The combined use of these three operators types (`XOR`, `AND`, `MULT`) in textual constraints gives the possibility of

modelling all alternative and complex geometry types (in the sense defined by ISO and Open GIS Consortium standards). This includes the types of geometries issued from spatial aggregation or composition; see [1] for information about spatial type standards in geomatics, and [13] for details concerning the expression of the aggregation with the three operators defined above.

It appears important to bring out an easy to use UML profile dedicated to GIS design, and completely based on standard extension mechanisms. In order to meet this need, the next subsection proposes a precise formalization of our UML extension in using stereotypes or tagged values.

### 2.2 Stereotype and Tagged Value Definition

**Geographic Data Type.** Geographic class instances are objects having a specific attribute named `geometry`. A class that specializes a geographic class is also geographic. A stereotype `<<geographic>>` is associated to each geographic class. We define that `BasicGeoType` is a type that generalizes the three types `point`, `polyline`, `polygon`. The type of the `geometry` attribute is an unordered collection. This collection is more precisely a bag (also called multiset) and can contain several geographic elements having a `BasicGeoType`. Thus, the type of a `geometry` attribute is *`Bag`*`(BasicGeoType)` i.e. type "bag containing elements that have the `point`, `polyline` or `polygon` type". This data structure based on bags corresponds to the perception of existing GIS data models by designers; indeed, to define a GIS, a total order on geographic objects is generally not needed and two objects can occupy the same XY location.

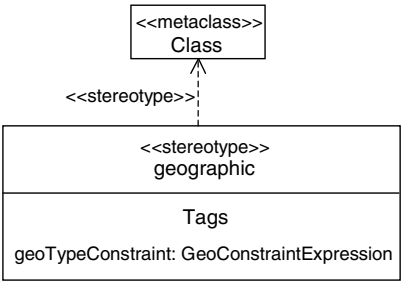



**Fig. 3.** Formalization for GIS design

**Type Constraint.** In order to easily define combinations of geographic types, this paper proposes to use a non-ambiguous textual representation of type constraints. In fact, constraints can be set on the contents of bags. Indeed, in using three operators (`XOR`, `AND`, `MULT`) and the geographic type names (`point`, `polyline`, `polygon`), it is possible to express what kind of elements can be contained in bags (see section 2.1). In order to model a type constraint, a new tagged value called `geoTypeConstraint` is introduced inside geographic classes. For example, the tagged value "`{geoTypeConstraint=point XOR (polygon MULT (0..*))}`"

declared in the definition of a class `C`, expresses that the `geometry` attribute value of `C` can only contain "one point" or "from 0 to *n* simple polygons". In this case, the form of the `geometry` attribute values is *Bag*{}, *Bag*{$point_1$} or *Bag*{$polygon_1$,…, $polygon_n$} i.e. an empty bag, a bag containing a point, or a bag containing from 1 to *n* polygons. Figure 3 formalizes the proposed profile in using UML notations. The expression type that combines `XOR`, `AND`, `MULT` is a `GeoConstraintExpression` which corresponds to the tagged value type. The next subsection illustrates the introduced concepts in the domain of agricultural information systems.

### 2.3 Domain-Specific Example of Type Constraint Use: Case Study in Agriculture

In agriculture, the sewage sludge spreading is considered as a good way to recycle waste issued from sewage plants; this technique consists in depositing sludge directly on fields. This type of low cost practices gives the possibility not only to recover waste, but also to fertilize the ground. In spite of its numerous advantages,

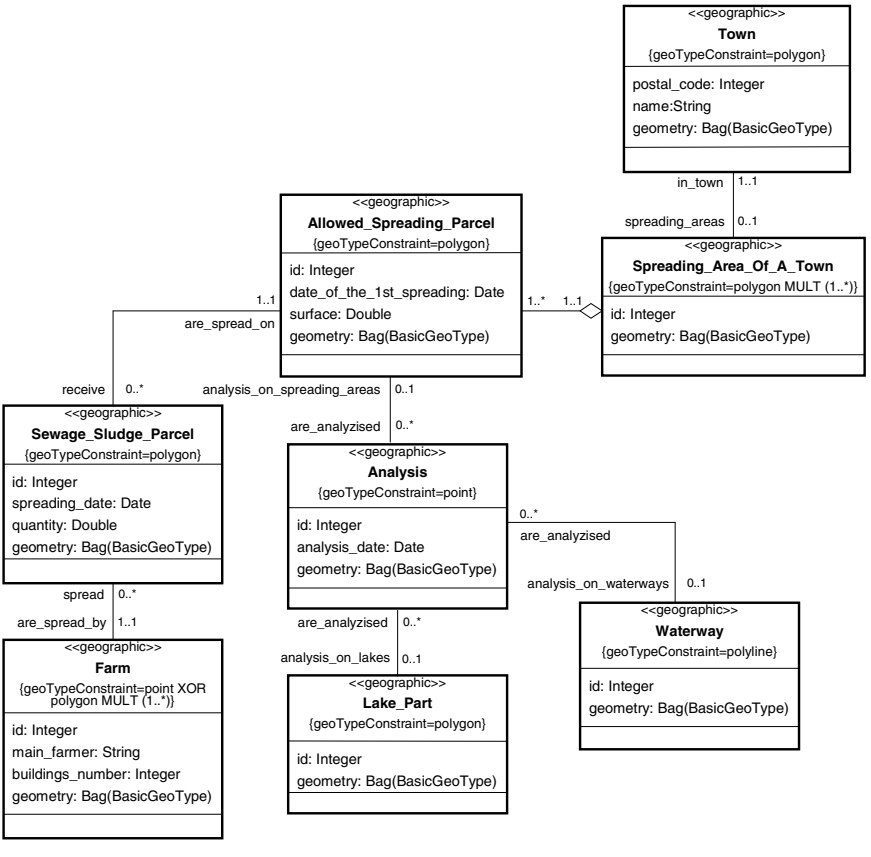



**Fig. 4.** Example of Sewage Sludge Spreading

the sewage sludge spreading must be monitored in order to avoid ground and waterway pollution. Indeed, too intensive practices could lead to an environmental deterioration. This could affect: a) areas that are close to the location where sewage sludge has been spread, and b) extended areas including hydrographical networks. This is the reason why a specific regulation has been defined e.g. for each town, allowed spreading areas must be defined in order to indicate precisely where sewage sludge could be spread without risk. To facilitate the monitoring of these activities, farms have to record areas where spreading had finally been carried out. The concentration of sewage must also be carefully monitored and governmental institutions usually organize ground and water analysis in different locations (lake, river,spreading area...).

The UML diagram of figure 4 corresponds to the modelling of object classes related to the implementation of sludge spreading monitoring inside a geographic information system. Farmers gradually spread sewage sludge inside "allowed spreading parcels" and at each farmer intervention, a new "sewage sludge parcel" is created; this parcel corresponds to a new area that received sludge. Thus, day after day, allowed spreading parcels are progressively covered by several sewage sludge parcels. Each instance of the "spreading area of a town" (SAT) class stores the "allowed parcels" aggregation of a town. Thus, for a town `t`, the geometry of a SAT is a bag including all the allowed parcel geometries occurring inside `t`. We also consider in this diagram two categories of analysis: a) water analysis of lakes and waterways, b) ground analysis of allowed parcels.

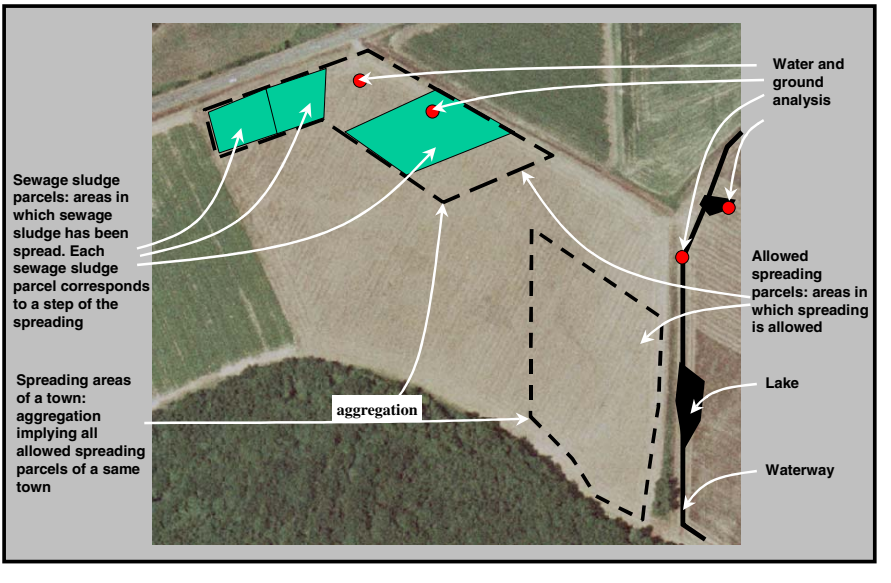



**Fig. 5.** Map Representation of Sewage Sludge Example

In the diagram, the geographic types are specified thanks to the stereotypes and the tagged values presented in this paper. Spatially speaking, a farm is a point or a set of polygons (each polygon is a building of the farm), depending on geographical representations available in the information system. Thus, `geometry` attribute values of the farm class are bags that include only one point or from one to several polygons. A sewage sludge parcel or an allowed spreading parcel is one polygon while each SAT geometry is a bag that can include several polygons (i.e. all allowed parcels geometries of a town). The spatial representation of a town is its geographical frontier. Waterways and lake parts are respectively modelled by polylines and polygons. The geographical location of an analysis is described by a point. The classes of the diagram are linked by classical UML associations according to natural structural relationships existing between objects in the information system.

In order to facilitate the understanding of this technical case study, figure 5 illustrates considered geographical objects on a map.

## 3 Extending OCL for Topological Constraint Expression

### 3.1 Introduction to OCL

The Object Constraint Language provides a framework for precisely defining constraints on a UML model in a formal way. OCL is textual and integrates several concepts issued from classical object-oriented languages. In the context of information systems, an important advantage of OCL is related to the fact that constraints are expressed at a conceptual level. OCL is used to specify invariant i.e. a condition that "must be true for all instances of a class at any time" [15].

In the following of the paper, we will consider that the information system in which the UML diagram of figure 4 will be implemented, will store only "good spreading practices". Thus, in order to illustrate OCL, different constraints defining the acceptable agricultural practices will be presented. For example, the following constraint requires that the surface area of allowed spreading parcels must be strictly greater than 100 square meters.

```
context Allowed_Spreading_Parcel inv:
 self.surface > 100
```

In this example, `self` is an instance of the `Allowed_Spreading_Parcel` class i.e. the class declared in the "context". OCL constraints must be "true" for each instance (i.e. each `self`) of the class specified in the "context".

More complex constraints can be built by using navigations along the associations between classes. In figure 4, an UML association is used in order to link sewage sludge parcels with the allowed spreading parcels receiving them. The next constraint illustrates the use of navigation in OCL by defining that the `date_of_the_1st_spreading` attribute of allowed parcels that are associated to sewage sludge parcels must be different from null.

Thus, `self` notation represents an instance of the `Sewage_Sludge_Parcel` class and the expression "`self.Allowed_Spreading_Parcel`" returns the `Allowed_Spreading_Parcel` instance associated to `self` by the relationship "`receive … are spread on`".

```
context Sewage_Sludge_Parcel inv:
 self.Allowed_Spreading_Parcel.date_of_the_1st_spreading<>''
```

As exemplified in the next constraint, names of association ends can also be used to navigate between classes; the semantics of this constraint is similar to the previous one.

```
context Sewage_Sludge_Parcel inv:
 self.are_spread_on.date_of_the_1st_spreading<>''
```

OCL provides several other functionalities for the definition of complex invariants. For example, the next expression specifies the following invariant: "the sum of sewage sludge quantities that are spread on an allowed parcel cannot exceed 100 units per square meter".

```
context Allowed_Spreading_Parcel inv:
 (self.Sewage_Sludge_Parcel.quantity->sum()/self.surface)<=100
```

The OCL expression "`self.Sewage_Sludge_Parcel.quantity->sum()`" provides the sum of quantities related to sewage sludge parcels that are associated to an allowed spreading parcel.

The next constraint means that an analysis must concern only one of these elements: one allowed spreading parcel, one waterway or one lake part. Note that "`self.C->size()`" provides the number of instances that are returned by the navigation "`self.C`".

```
context Analysis inv:
 (self.Allowed_Spreading_Parcel->size() + self.Waterway->size() +
  self.Lake_Part->size()) = 1
```

The `allInstances` construct can be applied on a class `C`; it returns the collection of all instances of `C`. Universal and existential quantifiers are denoted in OCL by `forAll` and `exists`. In the OCL syntax, `allInstances`, `forAll` and `exists` are prefixed operators. Logical implication can be also expressed by `implies`. The next constraint exemplifies these functionalities by specifying that each town has a proper postal code. Let `t` and `self` be two towns. If `t` and `self` are not the same town then their postal code must be different.

```
context Town inv:
 Town.allInstances->forAll(t|
       t<>self implies t.postal_code<>self.postal_code)
```

### 3.2 OCL Extension Definition

Among the range of spatial relations, this paper focuses more precisely on topological relations. To adapt OCL, new basic types are added, as presented in figure 6. The model of the figure extends a part of the OCL meta-model defined by [14]. Note that instances of classes in this meta-model are the types themselves, not instances of the domain they represent. Thus, three new basic geographic types generalized by `BasicGeoType` are added and each new type corresponds to a simple geographic type. In fact, each `geometry` attribute value (see section 2.2) is a bag of elements and each element in the bag has a `BasicGeoType` (i.e. `point`, `polyline` or `polygon`).

In order to describe topological constraints, new OCL operations are defined. In this paper, relations underlined by Egenhofer [4] have been chosen in order to illustrate our extension: overlaps, contains, is inside, are adjacent, covers, is covered, are disjoint, are equal. Each of these relations is considered between a pair of simple geometries (point, polyline, polygon). In some cases, it is possible to check topological relations between two simple geometries having a different type (for example between a point and a polygon). Figure 7 emphasizes the semantics of Egenhofer relations with two polygons.

The proposed OCL extension introduces eight operations for checking the described relations; one operation for each topological relation. These operations can appear in OCL expressions and can be applied between objects having the type `BasicGeoType` or the type *`Bag`*`(BasicGeoType)` i.e. the type "bag of elements that have a `BasicGeoType`". These operations are described in table 1 and the generic algorithm used for checking a topological constraint is defined in figure 8. Objects having a `geometry` attribute value equal to null are simply ignored by the algorithm. Indeed, null values often correspond to undefined or unknown data, and it seems preferable that this specific information doesn't negatively influence the topological constraint checking.

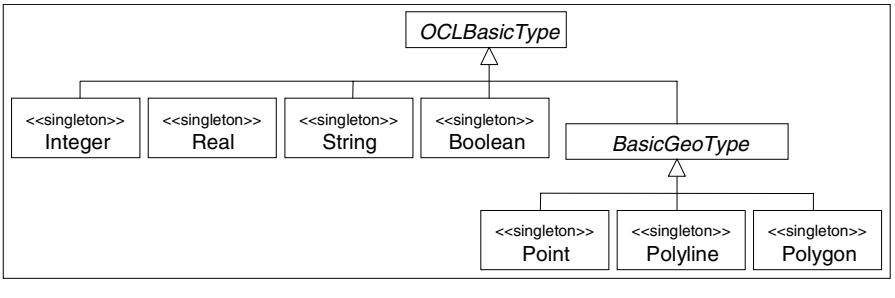



**Fig. 6.** New OCL Basic Types (point, polyline and simple polygon)

Concretely, the topologic operations presented in table 1 can have as parameters the `geometry` attribute (type *`Bag`*`(BasicGeoType)`) or one element of the `geometry` attribute (type `BasicGeoType` in the case of a geometry composed by several elements i.e. several parts). It is important to note that as presented in figure 8, if a parameter has the type *`Bag`*`(BasicGeoType)` and if its size is >1 then the

operations return systematically false; in other words, we only allow topological relations between pairs of simple element.

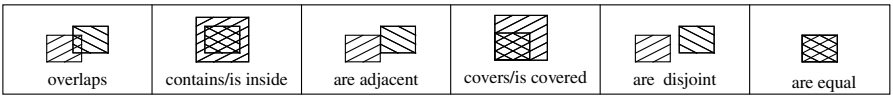



**Fig. 7.** Exemplification of Egenhofer Topological Relations with two Polygons

**Table 1.** New OCL operations for checking topological constraints

| Operations | Description |
|---|---|
| *g1*->overlaps(*g2*)<br>*g1*->contains(*g2*)<br>*g1*->isInside(*g2*)<br>*g1*->areAdjacent(*g2*)<br>*g1*->covers(*g2*)<br>*g1*->isCovered(*g2*)<br>*g1*->areDisjoint(*g2*)<br>*g1*->areEqual(*g2*) | We define that *g1* and *g2* are the parameters of the operations. The type of *g1* and *g2* is `BasicGeoType` or *Bag*(`BasicGeoType`). These operations return true or false (a boolean) depending on whether the topological relation between *g1* and *g2* is true or false. The generic definition of the operations is described in figure 8. |

Furthermore, elements of the `geometry` attribute can be accessed thanks to the standard OCL operations used on bags (`forAll`, `exists`, `select`, `reject`…). Indeed, these operations can be directly applied on `geometry` attributes.

For example, "for each element `e` in the `geometry` attribute value" is written as "`geometry->forAll(e|…)`". The type of `geometry` is *Bag*(`BasicGeoType`) and the type of `e` is `BasicGeoType`.

```
input: g1 and g2
output: true or false
//The topological relation can only be checked between two simple
//geometries. If g1 or g2 is a bag then it must contain only
//one element.
if   ( the type of g1 is Bag(BasicGeoType) and (size of g1)>1 ) or
     ( the type of g2 is Bag(BasicGeoType) and (size of g2)>1 )
then return false;
else {
//The topological relation is always true if one bag is empty.
     if ( the type of g1 is Bag(BasicGeoType) and (size of g1)=0 ) or
        ( the type of g2 is Bag(BasicGeoType) and (size of g2)=0 )
            then return true;
            else { if the potential type difference between
                   the simple geometries to compare
                   doesn't allow the topological relation checking
                   then return false;
                   else {
                          if the topological relation is
                          true between g1 and g2 then return true;
                          else return false; } } }
```

**Fig. 8.** Generic algorithm used by operations for checking if a topological relation is true or false

### 3.3 Spatial OCL in Practice

Thanks to some practical examples in the domain of agricultural information systems, this subsection points up the possibilities offered by the proposed OCL extension. This part of the paper continues the modelling of constraints based on figure 4 and related to "good agricultural practices".

**Spatial constraints between instances of different classes**. The next invariant defines that the geometry of sewage sludge parcels is inside the geometry of associated allowed parcels. The spatial relations "contains" and "covers" are used between the `geometry` attribute of geographic classes. The constraint presents a navigation from sewage sludge parcels to allowed spreading parcels and specifies that the "alphanumeric" association "`receive … are spread on`" existing in the information system between these two different types of parcels must imply containment or covering.

```
context Sewage_Sludge_Parcel inv:
 self.Allowed_Spreading_Parcel.geometry->contains(self.geometry)
  or
 self.Allowed_Spreading_Parcel.geometry->covers(self.geometry)
```

In the same way, the next constraint concerns the spatial containment of ground and water analysis inside lakes, waterways or allowed spreading parcels.

```
context Analysis inv:
 self.Allowed_Spreading_Parcel.geometry->contains(self.geometry)
  or
 self.Lake_Part.geometry->contains(self.geometry)
  or
 self.Waterway.geometry->contains(self.geometry)
```

The invariant "lakes and spreading parcels are disjoint" can be defined by the following clause. Unlike the previous constraint, this expression concerns all instances of two classes independently of an association between them. For each `Allowed_Spreading_Parcel` instance (denoted by `self`) and for each `Lake_Part` instance (denoted by `lp`), the geometry of `self` must be disjoint from the geometry of `lp`. Remember that in OCL, a constraint must be satisfied for each instance `self`. Thus, a "`forAll(self|…)`" is implicitly applied on the set of `Allowed_Spreading_Parcel` instances.

```
context Allowed_Spreading_Parcel inv:
 Lake_Part.allInstances()->forAll(lp|
       lp.geometry->areDisjoint(self.geometry))
```

**Spatial constraints between instances of a same class**. While previous examples underline topological relations between instances of different classes, the next OCL expression explores the possibility to declare spatial constraints on two instances of a same class. This example indicates that all allowed spreading parcels must be geographically disjoint.

```
context Allowed_Spreading_Parcel inv:
 Allowed_Spreading_Parcel.allInstances()->forAll(asp|
       asp.id<>self.id
         implies
       asp.geometry->areDisjoint(self.geometry))
```

**Spatial constraints implying sub-elements of geometry attributes**. As presented in the section 3.2, because the value of `geometry` attribute is a bag, the different elements of this attribute can be accessed thanks to the standard OCL operations used on bags (`forAll`, `exists`, `select`, `reject`…). The next example illustrates this possibility. The geometry of a SAT is the aggregation of all allowed spreading parcels related to a same town. Each SAT corresponds to a specific town. All `Allowed_Spreading_Parcel` instances defined in a same town are associated to the same `Spreading_Area_Of_A_Town` instance (see aggregation relationship presented in the diagram of figure 4). The `geometry` value of a SAT is a bag including the geometries of the associated `Allowed_Spreading_Parcel` instances. The next constraint shows that all elements of this bag must be spatially disjoint. The type of the `geometry` attribute is *`Bag`*`(BasicGeoType)` and consequently, the type of `p1` and `p2` is `BasicGeoType`.

```
context Spreading_Area_Of_A_town inv:
 self.geometry->forAll(p1,p2|
       p1<>p2 implies p1->areDisjoint(p2))
```

The farm geometry is a bag containing one point or from one to several polygons (depending on the representation available in the information system). This last constraint means that if a farm is represented by polygons then the `buildings_number` attribute must be equal to the number of elements composing the `geometry` attribute of the `Farm` object (i.e. the number of buildings composing the farm). In the example, the prefixed OCL operation `select(cond)` returns all elements of the `geometry` attribute value that satisfy the condition `cond`. Note that the OCL `oclIsKindof` function checks if the type of the element `t` is `Polygon`.

```
context Farm inv:
 self.geometry->select(t| t.oclIsKindof(Polygon))->size() > 0
         implies
       self.buildings_number = self.geometry->size()
```

## 4 Conclusion and Perspectives

In our work, we have naturally chosen UML and OCL because these languages became standard methods to model information systems. Firstly, we propose a complete formalism of concepts related to geographical type constraints. Secondly, existing UML-based formalisms only allow a topological constraint representation by relationships; thus, by integrating topological operations to OCL, we propose a language for expressing complex spatial constraints. Also, as presented in section 3,

the application of OCL operations like `forAll`, `select`, or `size` on the `geometry` attribute provides very precise constraint expressions on bags of geometries.

Thus, our work aimed at improving UML models thanks to geographical type constraints and topological constraints. In using the proposed formalism, when the designer writes the system specifications, the visual representation of geographic data illustrated in figure 5 becomes unnecessary because all spatial features are directly declared in UML diagrams and in OCL expressions.

Egenhofer topological relations have been used in our design of agricultural information systems but the proposed OCL extension can be easily adapted to new spatial relations (e.g. metric relations) in order to meet other domain needs. In this case, the GIS designer simply has to give the semantics of the new topological relations for pairs of geometries. As discussed in the following subsections, two main research perspectives will be studied regarding future work.

### 4.1 Perspective 1: Automatic Generation Inside GIS

An important challenge is to reduce the gap between the conceptual description of constraints and their implementation inside a GIS. This is the reason why we are interested in setting up a generator dedicated to the automatic generation of triggers from spatial OCL constraints.

In order to achieve this aim, we currently investigate the possibility to extend an existing tool named OCL2SQL in order to produce a mechanism for integrity checking in geographical databases. The open source OCL2SQL program is a powerful generator developed by [2], [3]; it offers the capability to generate automatically from an OCL expression *c*, a SQL query selecting all data that don't satisfy *c*. Once integrated inside a trigger (on data insertion, deletion and update), the query provides guards that guarantee the consistency of databases. Indeed, during each data update, the trigger checks if the generated SQL query returns tuples; if it's not the case then the update is accepted, else data modification is rejected. By this technique, it becomes, for example, impossible to insert data that violate a constraint.

We started to partially include our spatial extension to OCL2SQL by adding the new proposed OCL syntax and by studying the automatic generation for the Spatial SQL supported by Oracle Spatial. The next example illustrates a possible conversion of an OCL expression into Spatial SQL.

OCL constraint:

```
context Allowed_Spreading_Parcel inv:
 Allowed_Spreading_Parcel.allInstances()->forAll(asp|
       asp.id<>self.id
         implies
       asp.geometry->areDisjoint(self.geometry))
```

Spatial SQL clause selecting data that don't satisfy the OCL constraint:

```
SELECT * FROM Allowed_Spreading_Parcel self
WHERE EXISTS
( (SELECT id FROM Allowed_Spreading_Parcel)
  MINUS
```

```
 (SELECT id FROM Allowed_Spreading_Parcel asp
WHERE asp.id=self.id OR
      MDSYS.SDO_RELATE(asp.geometry, self.geometry,
      'mask=DISJOINT querytype=WINDOW')='TRUE') )
```

As illustrated by this example, Spatial OCL is an excellent alternative to write triggers. Without the use of OCL, the direct writing of queries associated to triggers seems to be a difficult task because the first role of SQL is not to model constraints but to select data; this explains the absence of the `forAll` notation in SQL whereas this notation can be directly used in OCL.

The desired architecture is schematized in figure 9. The new version of OCL2SQL will take as parameters: a) files containing spatial constraints, b) a XMI file corresponding to the UML conceptual schema of the database.

In the future, it could be also possible to consider automatic generation for other GIS target platforms.

Other complementary researches at Cemagref are also directed towards the generation of logical models from UML diagrams improved by stereotypes for the geographic type description [10].

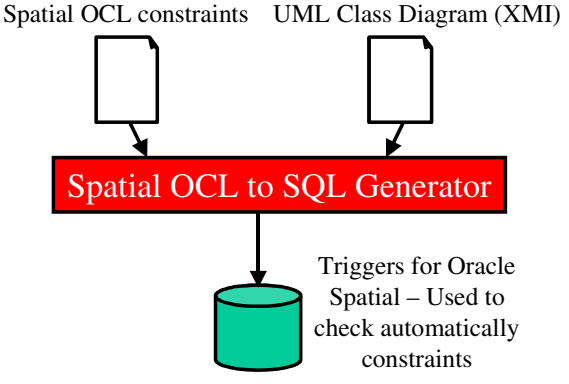



**Fig. 9.** Spatial OCL to SQL Generator

### 4.2 Perspective 2: Definition of More Complex Spatial Relations

Finally, we also study the modelling of constraints that imply more complex spatial relations; for example spatial relations on more than two simple geometries. A possibility could be to use Spatial OCL as a meta-"constraint language" in order to define precisely other relations. In this case, complex spatial relations could be formalized directly with Spatial OCL and their OCL definition could be associated to a function name with different parameters. Then the new functions could be used by designers in other OCL constraints. This research topic is closely linked to the knowledge of the Spatial OCL expressive power.

## References


1. Brodeur J., Bédard Y., Proulx M.J.: Modelling Geospatial Application Databases using UML-based Repositories Aligned with International Standards in Geomatics. Proc. of the Int. ACM Symposium on Advances in Geographic Information Systems, USA (2000) 39-46
2. Demuth B., Hußmann H., Loecher S.: OCL as a Specification Language for Business Rules in Database Applications. Proc. of the Conference on the Unified Modelling Language, USA (2001) 104-117
3. Demuth B., Hußmann H.: Using UML/OCL Constraints for Relational Database Design. Proc. of the Conference on the Unified Modelling Language, USA (1999) 598-613
4. Egenhofer M., Franzosa R.: Point-Set Topological Spatial Relations. Int. Journal of Geographical Information Systems, Vol.5(2). (1991) 161-174
5. Friis-Christensen A., Tryfona N., Jensen C.: Requirements and Research Issues in Geographic Data Modeling. Proc. of the Int. ACM Symposium on Advances in Geographic Information Systems, USA (2001) 2-8
6. Kösters G., Pagel B., Six H.: GIS-Application Development with GeoOOA. Int. Journal of Geographical Information Science, Vol.11(4). (1997) 307-335
7. Laurini, R.: Information Systems for Urban Planning. Taylor & Francis (2001) 308p
8. Lbath A., Pinet F.: Towards Conceptual Modelling of TeleGeoProcessing Applications. Proc. of the Int. Symposium on TeleGeoProcessing, France (2000) 25-39
9. MapInfo Corporation: MapInfo Professional user's guide, USA (2003)
10. Miralles A.: GIS Profile for Objecteering. Cemagref (2004)
11. OMG: Unified Modeling Language. OMG Specification (2001) 556p
12. Parent C., Spaccapietra S., Zimanyi E.: Spatio-Temporal Conceptual Models: Data Structures + Space + Time. Proc. of the Int. ACM Symposium on Advances in Geographic Information Systems, USA (1999) 26-33
13. Pinet F., Lbath A.: Semantics of Stereotype for Type Specification: Theory and Practice. Proc. of the Int. Conference on Conceptual Modeling (ER'01). Lecture Notes in Computer Science Vol.2224. Springer Verlag, Japan (2001) 339-353
14. Richters M., Gogolla M.: A Metamodel for OCL. Proc. of the Conference on the Unified Modelling Language, USA (1999) 156-171
15. Schmid B., Warmer J., Clark T.: Object Modeling With the OCL: The Rationale Behind the Object Constraint Language. Springer Verlag (2002) 281p


## Appendix: Semantics of Geographical Type Constraints

### A.1 Notations

A set is denoted by $\{element_1, \ldots, element_n\}$. In a set, any element can be represented only once. A bag is denoted by $Bag\{element_1, \ldots, element_n\}$. A bag is similar to a set, but it can contain duplicate element. The empty bag is $Bag\{\}$. The merge of bags is allowed by using the union operator. For example, $Bag\{a,b\} \cup Bag\{b,c\} \cup Bag\{\} = Bag\{a,b,b,c\}$. The bag combination operation (denoted by $\underline{\times}$) is a concept inspired by the cross-product; it provides the capability for combining sets of bags. Let $S_1$ and $S_2$ be two sets of bags, $S_1 \underline{\times} S_2 = \{ Bag_i \cup Bag_j \mid Bag_i \in S_1$ and $Bag_j \in S_2 \}$. For example, $\{ Bag\{a,b\}, Bag\{c\} \} \underline{\times} \{ Bag\{c\}\} = \{ Bag\{a,b,c\}, Bag\{c,c\} \}$. The bag combination

operation is associative. Also, because the bag is an unordered collection, this combination operation on sets of bags is commutative. $S^n$ is the bag combination operation of $S$ with itself $n$ times; if applied to a set of bags, $S^0 = \{Bag\{\}\}$ ; $S^1 = S$ ; $S^2 = S \underline{\times} S$ ; $S^3 = S \underline{\times} S \underline{\times} S$ ; ...

### A.2 Semantics

The semantics of a geographical type constraint associated to a class `C` can be described by the value domain related to the `geometry` attribute of `C`. A value domain of a `geometry` attribute is a set of bags and consequently, the value of a `geometry` attribute is a bag. *Dom*(`point`), *Dom*(`polyline`) and *Dom*(`polygon`) are respectively the infinite sets of points, polylines and polygons. For example, *Dom*(`polygon`) is the set of all the polygons that can be defined i.e. all possible polygons.

More precisely, we define that:

$$Dom(\texttt{point}) = \{\ Bag\{point_1\}\ ,\ Bag\{point_2\}\ ,\ \ldots\ ,\ Bag\{point_i\}\ ,\ \ldots\ \}$$
$$Dom(\texttt{polyline}) = \{\ Bag\{polyline_1\}\ ,\ Bag\{polyline_2\}\ ,\ \ldots\ ,\ Bag\{polyline_i\}\ ,\ \ldots\}$$
$$Dom(\texttt{polygon}) = \{\ Bag\{polygon_1\}\ ,\ Bag\{polygon_2\}\ ,\ \ldots\ ,\ Bag\{polygon_i\}\ ,\ \ldots\ \}$$

**Definition 1**

**Let $Dom_{Mult}$ be a function that applies a multiplicity '*min..max*' to a set of bags *S*:**

$$Dom_{Mult}\ (\mathrm{S}\ ,\ \text{'}min..max\text{'}) = S^{min} \cup .. \cup S^{k} \cup .. \cup S^{max} \text{ with } min \le k \le max$$

□

For example, $Dom_{Mult}$ (*Dom*(`polygon`) , '`0..2`') applies the multiplicity '`0..2`' to the set of possible polygons.

$Dom_{Mult}$ (*Dom*(`polygon`) , '`0..2`')

$= Dom(\texttt{polygon})^0 \cup\ Dom(\texttt{polygon})^1 \cup Dom(\texttt{polygon})^2$

$= \{\ Bag\{\}\ \} \cup \{\ Bag\{polygon_1\},\ldots,\ Bag\{polygon_i\},\ldots\ \} \cup$
$(\ \{\ Bag\{polygon_1\},\ldots,\ Bag\{polygon_i\},\ldots\ \} \underline{\times}$
$\{\ Bag\{polygon_1\}\ ,\ldots,\ Bag\{polygon_j\},\ldots\ \}\ )$

$= \{\ Bag\{\}, Bag\{polygon_1\},\ldots,\ Bag\{polygon_i\},\ldots\ ,$
$Bag\{polygon_1, polygon_1\},\ldots,\ Bag\{polygon_i, polygon_j\},\ldots\ \}$

Intuitively, the application of the multiplicity '`0..2`' on *Dom*(`polygon`) returns the value domain composed of bags that contain from 0 to 2 polygons.

**Definition 2**

Let *TC* be the infinite set of all type constraints i.e. the infinite set that contains all possible values of the `geoTypeConstraint` tag. We will define the function $Dom_{TC}$ that associates a type constraint to its value domain. The function $Dom_{TC}$ is defined by:

***R1.*** $Dom_{TC}(r) = Dom(r)$ if $r \in$ {`point`, `polyline`, `polygon`}

***R2.*** $Dom_{TC}(r \texttt{ AND } s) = Dom_{TC}(r) \underline{\times} Dom_{TC}(s)$

***R3.*** $Dom_{TC}(r \texttt{ XOR } s) = Dom_{TC}(r) \cup Dom_{TC}(s)$

***R4.*** $Dom_{TC}(r \texttt{ MULT } m) = Dom_{Mult}(Dom_{TC}(r), m)$

□

$Dom_{TC}$ provide the semantics of type constraints i.e. the corresponding value domains. Let *r*, *s* be two type constraints included in *TC*; *r* and *s* have exactly the same semantics iff $Dom_{TC}(r) = Dom_{TC}(s)$. The function $Dom_{TC}$ decomposes the value domain of a constraint into combination operations and unions of bags that contain only one element. While the bag combination operation corresponds to a conjunction between value domains, the union is a disjunction of value domains.

For example,

$Dom_{TC}$ `( (point XOR polygon) AND polygon AND (polyline MULT (0..2)) )`

$= (Dom(\texttt{point}) \cup Dom(\texttt{polygon})) \underline{\times} Dom(\texttt{polygon}) \underline{\times} (Dom(\texttt{polyline})^0 \cup Dom(\texttt{polyline})^1 \cup Dom(\texttt{polyline})^2)$

$= (\{Bag\{point_1\}, \ldots, Bag\{point_i\}, \ldots\} \cup \{Bag\{polygon_1\}, \ldots, Bag\{polygon_j\}, \ldots\}) \underline{\times} \{Bag\{polygon_1\}, \ldots, Bag\{polygon_j\}, \ldots\} \underline{\times} (\{Bag\{\}\} \cup \{Bag\{polyline_1\}, \ldots, Bag\{polyline_p\}, \ldots\} \cup \{Bag\{polyline_1, polyline_1\}, \ldots, Bag\{polyline_p, polyline_q\}, \ldots\})$

$= \{ Bag\{ point_1, polygon_1\}, \ldots, Bag\{ point_i, polygon_j\}, \ldots,$
$Bag\{ polygon_1, polygon_1\}, \ldots, Bag\{ polygon_j, polygon_k\}, \ldots,$
$Bag\{ point_1, polygon_1, polyline_1\}, \ldots, Bag\{ point_i, polygon_j, polyline_p\}, \ldots,$
$Bag\{ polygon_1, polygon_1, polyline_1\}, \ldots,$
$Bag\{ polygon_j, polygon_k, polyline_p\}, \ldots,$
$Bag\{ point_1, polygon_1, polyline_1, polyline_1\}, \ldots,$
$Bag\{ point_i, polygon_j, polyline_p, polyline_q\}, \ldots,$
$Bag\{ polygon_1, polygon_1, polyline_1, polyline_1\}, \ldots,$
$Bag\{ polygon_j, polygon_k, polyline_p, polyline_q\}, \ldots \}$

According to the previous type constraint, the value of a geometry attribute must include:

- (one point and one polygon)
- or (two polygons),

– or (one point and one polygon and one polyline),
– or (two polygons and one polyline),
– or (one point and one polygon and two polylines),
– or (two polygons and two polylines).

Two type constraints that are syntactically different can have exactly the same semantics. For example, this is the case for the next type constraints. The function $Dom_{TC}$ allows to determine the semantics equivalence.

*geoTypeConstraint1=*
```
(point MULT(0..1)) AND ((polygon XOR point) MULT(0..1))
```

*geoTypeConstraint2=*
```
((point MULT(0..1)) AND (polygon MULT(0..1)))
XOR (point MULT(0..2))
```

Thus, $Dom_{TC}$ (*geoTypeConstraint1*) = $Dom_{TC}$ (*geoTypeConstraint2*)
= { $Bag\{\}$, $Bag\{ point_1 \}$, … , $Bag\{ point_i \}$, … ,
$Bag\{ polygon_1\}$, … , $Bag\{ polygon_j\}$, … ,
$Bag\{ point_1 , point_1\}$, … , $Bag\{ point_i , point_j\}$, … ,
$Bag\{ point_1 , polygon_1\}$, … , $Bag\{ point_i , polygon_j\}$, … }

# Location and Tracking Services for a Meta-UbiComp Environment

Antonio Coronato[1] and Giuseppe De Pietro[2]

[1] DRR-CNR, Via Castellino 111, 80131 Napoli, Italy
coronato.a@ na.drr.cnr.it
[2] ICAR-CNR, Via Castellino 111, 80131 Napoli, Italy
depietro.g@cps.na.cnr.it

**Abstract.** Current prototypes of UbiComp environments are bounded to a physical site equipped with a WLAN. However, next generation of UbiComp environments will have to aggregate different physical sites, spread over a wide geographic area, each one equipped with an own WLAN, and interconnected by the internet. The emerging model is a meta-environment that integrates different physical environments. It must provide users with a uniform interaction model independently from the physical site they are in. Clients, active in a site, have to get access only to services available in that site. Moreover, users, who move from one site to another one, must have the possibility of suspending their computations before leaving the site and of resuming them once in the new site. These needs call for advanced location and tracking services. This paper presents a location and tracking service for meta-UbiComp environments. The location function is in charge of determining mobile clients active inside the meta-environment, at every time, in every physical site. Mobile users get access only to services available in the physical site they are in. In addition, users are tracked, and the environment automatically reconfigures itself when they move from one location to another one, or when they definitively leave the environment. This makes the environment able to reliably handle resources and services.

## 1 Introduction

Technology evolution in hardware design and development (including microprocessors, network bandwidth, and wireless devices), which has led to a dramatic improvement in terms of performance to cost ratio and of the integration scale factor, is leading towards the implementation of the Mark Weiser vision [1]. The emerging computing model is called *Ubiquitous Computing* (UbiComp). *Ubiquitous Computing* inherits characteristics from the *Mobile Computing* [2], which is fundamentally related to accessing computing services independently from physical user movements. However, the concept of *Ubiquitous Computing* extends the model of *Mobile Computing* with an interaction model which allows the device to i) receive information from the environment about available services, ii) dynamically configure end-points for existing services, and iii) build new

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 179 – 191, 2004.

services, which are to be customized according to specific user needs, as well as to environment and device specific characteristics.

The ultimate goal is the development of environments where highly heterogeneous hardware and software components can seamlessly and spontaneously interoperate, in order to provide a variety of services to users independently of the specific characteristics of the environment and of the client devices [4]. Therefore, mobile devices should come into the environment in a natural way, as their owner moves, and transparently, that is owner will not have to carry out manual configuration operations for being able to approach the services and the resources.

Current realizations of Ubiquitous Computing environments are characterized by being bounded to a unique physical site [11][12][13]. Such environments are participated by a set of users, hardware, and software components, which is highly dynamic and cannot be predicted in advance. As a consequence, the sudden departure or arrival of a service, device, or user should be considered normal operation, not an exceptional condition or a failure requiring special handling [5]. This characteristic calls for location and tracking mechanisms. Such a need is extremely exacerbated for the next generation of UbiComp environments. Indeed, the Ubiquitous Computing model has to scale to large enterprises, which might have different offices distributed over a wide area, in different physical sites. In that case, the environment has to offer a unique interaction model to its users. In particular, users should access to the same set of services independently on the site, with the same interface. Services that require resources not available in all physical sites should be accessed only in those sites where the required resources are available. Moreover, users should be able to suspend their computation in a site and to resume it from a different one.

This requires base services for determining mobile users in the environment and their location. In addition, the environment must be able to track user movements. As a matter of fact, when a user is no longer active in a site, it is possible that he/she has definitively left the environment, or his/her device has failed, or he/she has left the physical site to reach another site, or more simply he/she has turned off his/her device to save battery or to have a break. Actually, since active devices affect the environment in terms of in use/available resources and services, we can conclude that the environment must reliably handle resources and services when users move around. It must be able to predict whether the user will require to resume its computation later or not. Then, active resources must be kept allocated or freed.

This paper describes the implementation of a dependable location and tracking service for a Meta-UbiComp Environment, that is a logic environment composed by different physical sites (i.e. local environment). From now on, with the term environment we will refer to the concept of meta-environment. It is also worth to clarify that as location system, we mean a mechanism able to detect active devices in a particular location at every time [3], and we identify a location with a physical site equipped with wireless access points; thus, we are not interested in active location systems (like those presented in [6]) that provide information about the position of a device within a physical site. The location information are used to

track user movements. Then, higher level functionalities infer the user intentions depending on his/her movements inside the environment and on the services he/she has activated. Finally, specific strategies are presented to make the environment able to reliably handle resources and services while users move inside the environment.

The rest of the paper is organized as follows. Section 2 presents the target scenario. Section 3 describes the location and tracking services. Section 4 presents the services architecture. Section 5 concludes the paper.

## 2 Meta-UbiComp Environments

### General Model

The physical model of a Meta-UbiComp environment is shown in figure 1. The environment consists of several sites (local environments) interconnected each other via the internet. Every site is equipped with wireless access points and other resources. Several user services are available in the environment. Table 1 shows an example of services availability for the different physical sites of the environment. A service is available in a specific site if the resources it requires are available there; or it runs in a different site, but it can present results in other sites.

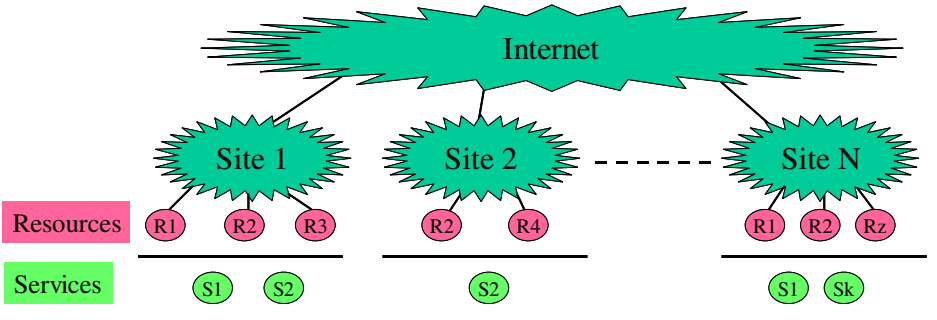



**Fig. 1.** Wide area UbiComp model

**Table 1.** Services availability

| Service | Required resources | Availability |
|---|---|---|
| S1 | R1, R2 | Site1, Site N |
| S2 | R2 | Site 2 |
| Sk | Rz | Site N |

**A Real Meta-environment**

The real environment we analyze consists of two sites. Site 1 is located in Naples. It is equipped with a printer and other resources. Site 2 is located in a suburb quarter, a few kilometers far from site 1. It is equipped with a multimedia laboratory and some multimedia rooms. In particular, the following resources are available:

- Motion Capture System – This is a system for capturing human motions. It consists of several cameras, which capture movements performed by an actor, and a graphic station, which reproduces movements as a skeleton accordingly with actor's motions;
- Rendering Station – This is a workstation for rendering row motion data in 3D graphic applications;
- Projector – This is a projector, driven by a pc, to project multimedia presentations;
- Streaming Server – This server hosts a video streaming application;
- E-Testing Server – This server hosts an E-Testing application;
- Printers and other resources.

The environment supports the following application services:

- *MotionCaptureService* – This service drives the motion capture system. An actor (equipped with optical markers) moves around in the multimedia laboratory. Several cameras capture his movements that are reproduced on a graphic station. The graphic station shows a skeleton, which moves accordingly with the actor, and records data movement in a file. This service must be available only in site 2 because it requires that the actor move in the physical capture area of the system;

- *RenderingService* – This service enables users to submit row motion data and to build 3D graphic applications. This is a computational service that executes processes over the Rendering Station. However, it should be available in both sites. Indeed, since computation doesn't require user-service interactions, user can submit the input row motion file and choose the rendering options via a dialog form; successively, he/she can take up the output rendered file at the end of the job. Thus, although computation is performed by the rendering station, input file can be submitted from a remote place;

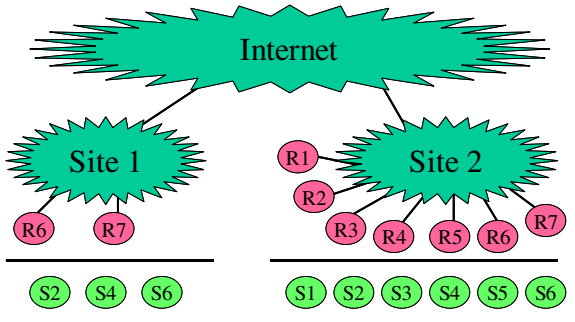
- *PresentationService* – This service enables a user to project its presentation in the multimedia room. The service receives a pdf/ppt file via a dialog form and then enables speaker to control the presentation flow. This is an interactive service, which requires the speaker to be in the room for presentation. As a consequence, the service must be available only in Site 2;

- *VideoConferenceService* – This service enables attendees to follow a presentation on their personal mobile device. A video server captures a presentation with its videocam and streams it over the network. The

service isn't interactive. Users receive presentation images, but do not interact with the speaker. This service must be available in all sites;

- *E-TestingService* – This service enables to perform on-line evaluation tests for courseware activities. When a session test starts, students must be in the multimedia room. Evaluation tests are synchronized and students have a predefined period of time for completing each test section. Students can interrupt their test by explicitly closing the service or by leaving the multimedia room. This service must be available only in the multimedia room of site 2;
- *PrintService* – This service enables users to print their pdf documents. This service must be available in both sites. However, the print service must use the local printer of the site in which user requires printing.

Figure2 shows the environment, the allocated resources, and the available services.




| Resources | |
|---|---|
| R1 | Motion Capture System |
| R2 | Rendering Station |
| R3 | Projector |
| R4 | Streaming Server |
| R5 | E-Testing Server |
| R6 | Printer |
| R7 | Other resources |

| Services | |
|---|---|
| S1 | MotionCaptureService |
| S2 | RenderingServices |
| S3 | PresentationService |
| S4 | VideoConferenceService |
| S5 | E-TestingService |
| S6 | PrintService |

**Fig. 2.** A real meta-UbiComp environment

We have already cited that users-environment interactions must take place spontaneously.

In this scenario, with regard to activities and movements that a user can perform, several events might occur. In particular, the following events can take place:

- a user appears in a site; this event occurs when the user enters in the environment and a list of available services, for that site, must be provided;
- a user disappears from a site; this event occurs when:
  i. the user definitively leaves the environment;
  ii. the user's device temporarily or permanently fails;
  iii. the user turns off his/her device to have a break or to save battery, but he/she wants to resume computation later and/or in a different location;
  iv. the user moves through a "black" zone, that is an area of the physical site not covered by wireless connectivity.

Such events call for handling mechanisms. In fact, once a mobile user enters in a site, the environment has to locate him/her and to provide the list of available services for that site. Moreover, once a user disappears from a site, the environment has to infer whether he/she is going to come back later or not. In other words, a user could disappear from a site in the middle of a computation. Firstly, this event has to be recognized by the environment. Then, the environment has to decide either to free allocated resources or to keep them active in the environment. The decision must be taken depending on the services activated by the disappearing user. In fact, with respect to our scenario, if a user turns off his/her device after having submitted a row data file for rendering, probably he/she will come back (reappear) in the environment to pick up results. Differently, if a student gives up an evaluation test session, the environment has to free the allocated resources because the student can not be admitted anymore to the same session. In general, we can classify services in highly interactive services (HIS) and lowly interactive services (LIS) (or computational). When a user interrupts a highly interactive service, probably he/she will not come back to resume. Differently, if a user disappears from the environment when computational services are running, probably he/she will come back later to pick up results.

In our scenario, the services are classified as HIS or LIS as reported in table 2.

**Table 2.** User services availability

| Service | Required resources | Kind of Service | Availability |
|---|---|---|---|
| MotionCaptureService | Motion Capture System | HIS | Site2 |
| RenderingService | Rendering Station | LIS | Site 1, Site 2 |
| PresentationService | Projector | HIS | Site 2 |
| VideoConferenceService | Streaming Server | LIS | Site 1, Site 2 |
| E-TestingService | E-Testing System | HIS | Site 2 |
| PrintService | Printer | LIS | Site 1, Site 2 |

## 3 The Location and Tracking Services

Location discovery mechanisms are not supported by traditional distributed environments, which typically are not able to recognize user disconnections, or when they do that, they always treat disconnections as system failures. Differently, modern ubicomp environments must be equipped with location discovery systems. However, earlier implementations limit the service responsibilities to identifying active objects into a physical area or site.

Our location service is able to locate active mobile users in all the physical sites of the environment. Most important, the location service has been coupled to a tracking service, which enables the environment to infer user intentions and to reliably handle allocated resources. Such services operate in a distributed environment, which is

composed by two distinct sites, equipped with wireless access points, and interconnected by the internet. Currently, the environment is not equipped with active location systems like active badges or sensors.

In order to locate active devices and to reliably handle user disconnections, the environment must be able to distinguish among the events reported in table 3. Table 3 also reports main requirements for the environment in order to achieve spontaneous and transparent interactions. Indeed, while Event 1 requires to provide the incoming user with network connectivity, Events 2 to 5 require specific disconnection strategies. Moreover, events 2 to 5 all manifest themselves identically (the user's device becomes unavailable in the environment); but, they require different reactions of the environment. Indeed, events 2 and 3 require that the allocated resources be freed, whereas, events 4 and 5 require that the allocated resources continue to be active.

Therefore, the environment must support proper connection, disconnection, and tracking strategies.

**Table 3.** Events and required behavior

| | Event | Required behavior |
|---|---|---|
| 1 | A new user comes into the environment | The environment provides network connectivity on-the-fly without any manual configuration action |
| 2 | The user definitively leaves the environment | The environment frees allocated resources |
| 3 | The user's device definitively fails | The environment frees allocated resources |
| 4 | The user's device temporarily fails or the user's device is temporarily unavailable in the environment | The environment temporarily suspends computation and is ready to resume |
| 5 | The user turns off his/her device to have a break or to save battery, but he/she wants to resume computation later and/or from a different site | The environment saves computation state and let user to resume later |

**Connection Strategies**

The mechanism we chose for providing network connectivity on-the-fly is based on the Dynamic Host Configuration Protocol (DHCP) [8], which is a well known solution for the implementation of basic location functions [7]. DHCP dynamically assigns an IP address to an incoming device, that is, then, able to access to the network. This realizes the desired behavior for event 1 without making particular assumptions about the client device, except that it must be DHCP enabled. However, standard DHCP has not been devised for highly dynamic environments. As a matter of fact, the IP address provided to the entering device is locked until

the lease time expires. The lease time is a parameter that typically varies from 12 to 24 hours. During this time, standard DHCP doesn't take care about possible early user disconnections. Some limitations of the standard DHCP in mobile computing were already pointed out in [9] and are only partially faced by the forthcoming DHCP RFC [10] that introduces the possibility of forcing the mobile device to renew the IP request once the lease time (for the old IP address) expires. However, this isn't enough for UbiComp environments. For this reason, some additional functions must be developed. In particular, checkpointing activities must be performed to recognize when devices are no longer active inside the environment. Moreover, the DHCP must be able to release the network allocated resources on-demand.

### Disconnection Strategies

Events 2 to 5 in table 1 require disconnection strategies. In particular, such events require specific behaviors for the environment, which must be able to distinguish these events. To solve this problem, we classified services as Lowly Interactive Services and Highly Interactive Services. This classification enables the environment to infer the client status once its device is no more traceable into the environment. In particular, in the case of a LIS active, if the user device becomes unavailable, most likely event 5 occurred but, in any case, the strategy is to free the IP address immediately and to continue computation until the results are obtained, then a timer starts. Computational resources are freed as soon as the user comes back to the environment and picks up results (figure 3.a) or the timeout expires (figure 3.b).

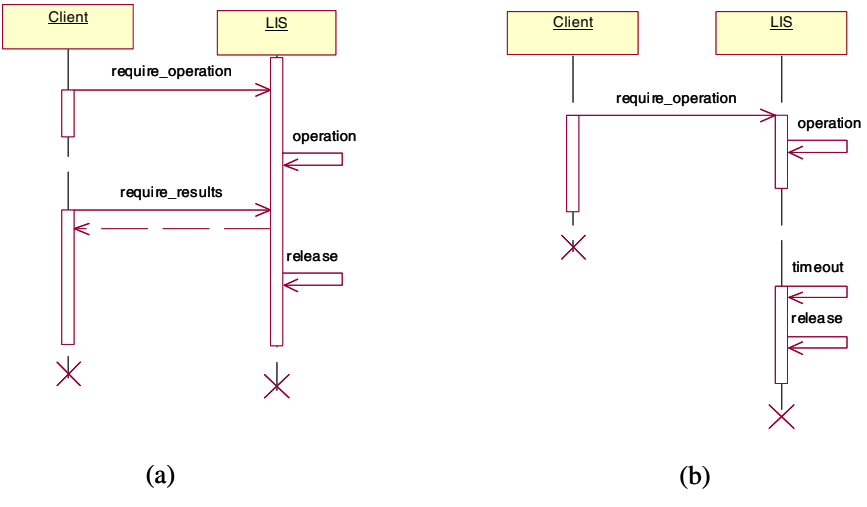



**Fig. 3.** Disconnection strategies for Lowly Interactive Services

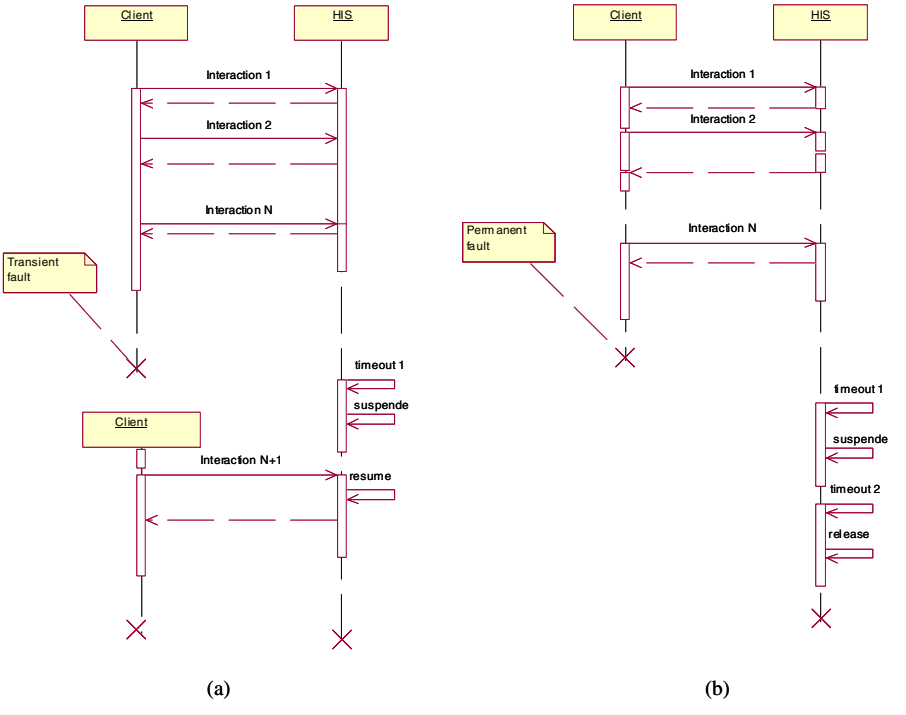



**Fig. 4.** Disconnection strategies for Highly Interactive Services

Differently, in the case of an HIS active, once the user's device gets unavailable, event 5 can be exclude (generally, no student would turn off his device in the middle of an evaluation test to have a break as well as no speaker would turn off his device during his presentation to save battery). At this time, the service must stop itself and wait for a timeout. If the user's device reappears in the environment before the timeout expires as in figure 4.a, likely event 4 occurred, whereas if the timeout expires as in figure 4.b, event 2 or 3 is assumed. In the case of event 4, the HIS must resume as the client gets available. In opposition, when event 2 or event 3 occurs, allocated resources must be freed and the service must release. In figure 2, timeout1 is the time the HIS must wait, once the user has got unavailable, before suspending itself. Timeout2 is the time the HIS must wait before releasing itself after having already suspended itself.

### Tracking Strategies

Generally, a user could activate more than one service in a session. In order to track user activities and movements, the state machine depicted in figure 5 can be adopted. Such a state machine consists of states and macro-states.

States:

- *LIS* – At least a LIS is operating. No HIS is active.
- *BC_WAIT* – The active LIS has completed its calculus and is waiting for the client who picks up results.
- *HIS* – At least a HIS is active. No LIS is active.
- *LIS_HIS* – At least one HIS and one LIS are active.
- *FC_WAIT* – At least a HIS is active and all active LISs have completed their calculus and wait for user who picks up results.

Macro-states:

- *BackgroundComputing* – In this macro state only LISs are active.
- *ForegroundComputing* – In this macro state at least a HIS is active.
- *NoComputing* – Neither LISs nor HISs are active.

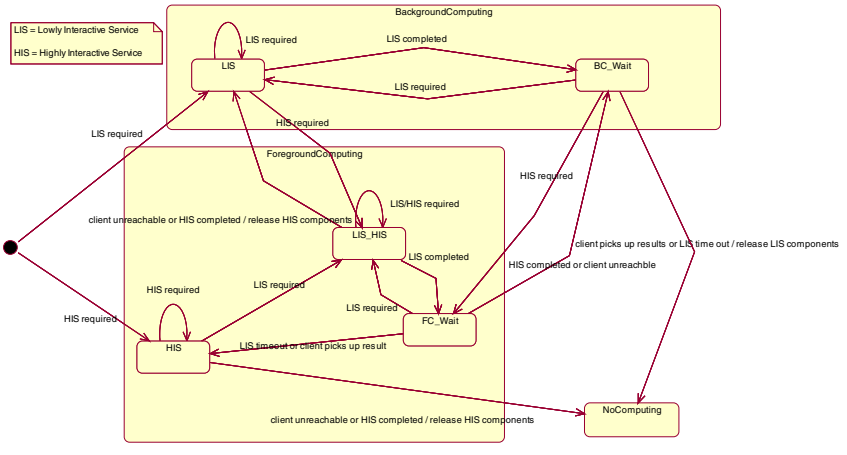



**Fig. 5.** Client state machine

## 4 Implementation Details

We built a tracking service and a two-levels location service. The tracking service is unique for all the sites of the environment. Differently, the location service is composed by a global location service, which is unique for the environment, and several site location services, one per site.

The services architecture is described in figure 6. It consists of the following components:

- *DHCPService* – This component implements a DHCP service. It provides network connectivity to the incoming devices as a standard DHCP, but it has additional functionalities. In particular, it communicates to the *SiteLocationService* when new devices are active in the site. Moreover, it

can release allocated IP addresses on-demand. Such a characteristic enables the *TrackingService* to require to free allocated network resources when a disappearing user is supposed to definitively leave the environment. Each site needs a local *DHCPService*.

- *EcoService* – This component implements ping functions in order to establish which devices are still active in the site. The *EcoService* is activated and deactivated by the *TrackingService*. Each site needs a local *EcoService*.
- *SiteLocationService* – This component handles a list of devices active in the site. It directly interacts with the *GlobalLocationService* and the *TrackingService*. It also forwards messages, coming from the *GlobalLocationService* and the *TrackingService*, towards the local *EcoService* and the local *DHCPService*. Each site needs a local *SiteLocationService*.
- *GlobalLocationService* – This component handles mobile devices location in the environment. It interacts with *SiteLocationServices* for having information about local positons, and with the *TrackingService* to communicate changes of location.
- *TrackingService* – This component tracks users' activities and movements within the meta-environment. In particular, it handles a state machine like the one depicted in figure 5 for each user. As a consequence, this service infers user intentions and drives other services to keep resources active or to free them.

Inter-site components interacts each other over a CORBA platform, whereas intra-site components communicates over the internet using the SOAP protocol.

The application services, which we presented in section 2.2, support the interfaces shown in figure 7. This makes possible the interaction of the services with the location and tracking services.

When a new device comes into the environment, it dynamically obtains an IP address from the *DHCPService*. The *DHCPService* communicates to the *SiteLocationService* that a new device is active. The *SiteLocationService* activates an Eco function for the new device, updates its internal data structures, and communicates that a new device is active in its site to the *GlobalLocationService*, which forwards this information to the *TrackingService*. The *TrackingService* creates an instance of the state machine shown in figure 7 for the entering device, and updates it accordingly with user requests.

When the device becomes inactive, the *EcoService* communicates such a condition to the *TrackingService*, which requires to free allocated resources or to leave them active depending on the user state. While in the *ForegroundComputing* or *NoComputing* state, the *TrackingService* forces to free allocated resources. Whereas, in the *BackgroundComputing* state, the *TrackingService* keep resources allocated.

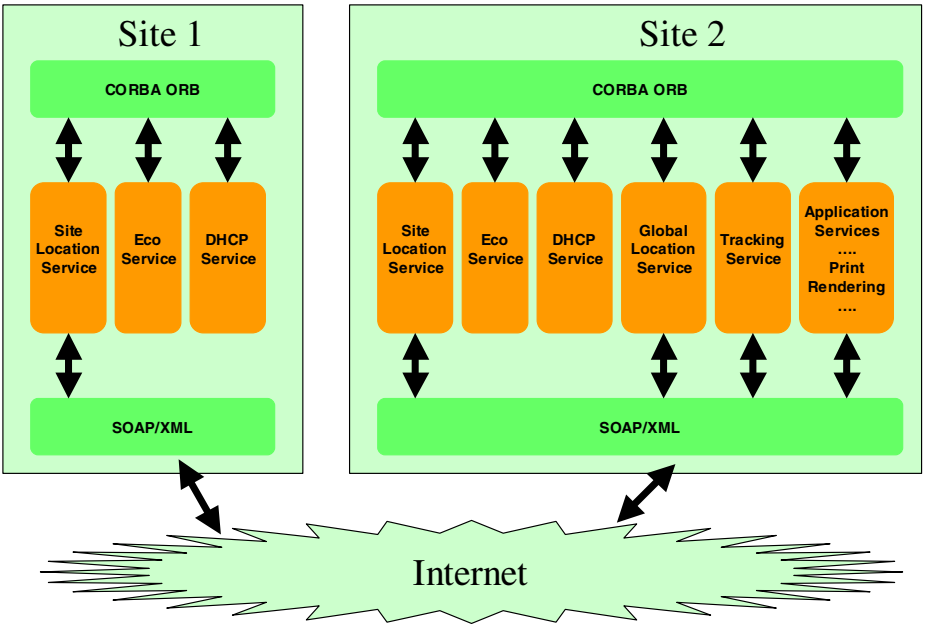



**Fig. 6.** Services architecture

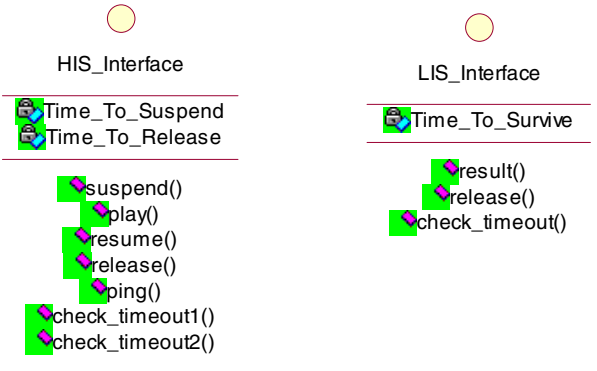



**Fig. 7.** Application services' interfaces

## 5 Conclusions and Directions for Future Works

Modern ubiquitous computing environments needs advanced location and tracking functions. In this paper we presented a couple of services for locating and tracking active mobile devices in a wide-area UbiComp environment. We also presented some strategies for making the environment able to reliably handle user disconnections.

Future work will aim to implement functions for handling a finer-grain concept of location. In particular, the environment will be equipped with active location systems. In that case a singular site will have different locations insight. This would make the environment able to improve its services. As an example, the print service could print a document at the user's nearest printer of the site.

In order to support such characteristics, we plan to introduce a new component in our architecture, namely the *LocationSystem*. A *LocationSystem* will be in charge of handling the list of mobile devices detected by a singular active location system. As a consequence, each location system will be driven by a specific *LocationSystem* component, and the list of mobile device active in a site will be obtained as logical sum of the lists handled by the *LocationSystems* of that site.

## References

[1] M. Weiser, "The Computer for the 21st Century", Scientific AM, September 1991, reprinted in IEEE Pervasive Computing, Jannuary-March 2002.
[2] K. Lyytinen and Y. Yoo, "Issues and Challenges in Ubiquitous Computing", Communications of ACM, December 2002, Vol. 45, N.12.
[3] S. Fischmeister, G. Menkhaus and A. Stumpfl, "Location-Detection Strategies in Pervasive Computing Environments", in the proc. of the 1st international conference on Pervasive Computing, PERCOM03.
[4] D. Saha and A. Murkrjee, "Pervasive Computing: A Paradigm for the 21st Century", IEEE Computer, March 2003.
[5] T. Kindberg and A. Fox, "System Software for Ubiquitous Computing", IEEE Pervasive Computing, January-March 2002.
[6] J. Hightower and G. Borriello, "Location Systems for Ubiquitous Computing", IEEE Computer, August 2001.
[7] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski, "Challenges: An Application Model for Pervasive Computing", in the proc. of the 6th ACM/IEEE Int. Conference on Mobile Computing and Networking, MOBICOM2000.
[8] R. Droms, "Dynamic Host Configuration Protocol", RFC 2131, Internet Engineering Task Force, www.ietf.org.
[9] C. E. Perkins and K. Luo, "Using DHCP with computers that move", Wireless Networks, 1995, pp 341-353, Baltzer AG, Science Publisher.
[10] Y. T'Joens, et all, "DHCP Reconfigure Extention", RFC 3203, Network Working Group, www.ietf.org.
[11] http://cobra.umbc.edu/
[12] http://oxygen.lcs.mit.edu/
[13] Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste P., "Project Aura: Toward Distraction-Free Pervasive Computing", IEEE Pervasive Computing, April-June 2002

# Applying Structural Computing Paradigms to Domain Analysis

## By Example of Knowledge Transfer in Higher Education

Armin Ulbrich[1], and Klaus Tochtermann[1,2]

[1] Know-Center Graz,
[2] Institute for Knowledge Management, Graz University of Technology,
Inffeldgasse 21a,
8010 Graz, Austria
{aulbrich, ktochter}know-center.at
http://www.know-center.at

**Abstract.** This paper deals with the application of research results from Structural Computing (particularly the grand unified theory) to the domain analysis process. The domain analyzed is knowledge transfer in higher education which includes supporting teaching, researching and learning. In order to create an appropriate knowledge transfer system, concepts of the domain in question are analyzed applying the grand unified theory. The "personal folder" qualifies as the most basic structuring mechanism for this application domain. The personal folder concept is investigated and its properties are discussed from the data, structure and behavior perspective.

## 1 Introduction

In higher education, researchers, students and lecturers have similar work practices concerning the acquisition and distribution of new and important knowledge. Normally, these work practices include steps such as search for resources (literature and tools) that are relevant in current working fields and contexts, collect and store them, classify them and make them available for discussion or for sharing them with peers and learners. A group of researchers at Graz University of Technology and Know-Center are currently designing a system for supporting the work practices outlined above. The objective is to develop a digital library environment that provides support in the domain knowledge transfer for higher education' in its broadest sense. This paper seeks answers to issues, which occur during the process of analyzing the problem domain.

The concept of the *personal folder* turns out to be the one distinguished structuring and structured entity that can be found throughout several functional areas of the domain in question. The personal folder is analyzed from several viewpoints that take especially its structuring functionality as well as its processing functionality into consideration. The application of this work to Meta-Informatics is actually two-fold. Firstly, there is a description of what we find to be the most essential or atomic under-

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 192 – 205, 2004.

lying abstraction that makes up and provides the structuring mechanism of a specific domain. Secondly, issues concerning the processing capabilities of the given abstraction are dealt with, that is, what are the processing operations that are executed upon the structural abstraction and what processing operations has the structural abstraction to provide. Since there has been an extended discussion on behavior in Structural Computing ([4]), particularly the last aspect appears to be quite relevant for the application of this work to Meta-Informatics. This discussion is considered to be of special importance in the case of the analysis and design of the personal folders.

This paper is structured as follows: In section 2 current research results from the field of dynamic personalization and Structural Computing are shown in order to position this paper in the context of related research areas. In section 3, first the analyzed domain is outlined briefly and then the basic underlying structuring concept is examined and discussed. Section 4 concludes with a brief discussion of the findings.

## 2 Positioning This Contribution Within Related Work

This contribution deals with domain analysis in the field of *knowledge transfer* tools for higher education. There are of course several aspects such as educational issues, system integration issues, performance and scalability issues and other that are heavily related to the work described in this paper. Although these issues are considered to be highly relevant, this contribution merely focuses on analyzing the domain by applying research results and paradigms from the fields of personalization in knowledge-based systems and Structural Computing. This section gives a brief outline of how our work is put in the context of work done in the fields mentioned.

### 2.1 Dynamic Personalization

Any person operating with a computer system is working within her own specific *user context*. This context might be defined by the person's identity, current activity, knowledge, competencies, her role within a given organization and many more factors. It has been shown ([8] and [9]) that especially knowledge-intensive work (i.e. execution of work activities that rely to a large extent on the creation or application of knowledge) needs to be supported by software tools that are able to adapt themselves depending on a user's specific context. Additionally, it has been shown ([5]) that there is a strong demand for software tools that take possible changes into consideration and allow for dynamically adapting not only content and features at run-time but also the respective *adaptation policies* in use. In analogy to the term user context, the status of a system as far as the content, features and active policies are concerned, is referred to as *system context*. The concept of considering context and adapting content and features accordingly is referred to as personalization, the additional consideration of change and dynamics is consequentially referred to as dynamic personalization.

[5] defines and describes a meta-model for a software system that provides dynamic personalization features. There are three fundamental entities within the system: The first entity is called UserContext; it represents the user operating with the system. The second entity represents content or system features which can be

personalized. This entity is referred to as PersonalizableEntity. The third entity is responsible for processing and performing dynamic personalization. It is referred to as PersonalizationEngine. The entities are related to one another as follows:

Whenever the user context changes (entity UserContext) the processing entity (PersonalizationEngine) is triggered to operate. This operation leads to the personalization of the target entity (PersonalizableEntity) that is the adaptation of the content or the system features.

Whenever the target entity (PersonalizableEntity) is altered for some reason, an event occurs and as a result of that the processing entity is triggered again and the user context is updated.

The processing entity relies on rule-based interpretation of conditions which change according to event. Occurrences of events trigger the processing entity to perform certain actions. The mechanism behind this is called event-condition-action (ECA). An event occurs whenever the user's context changes (e.g. location, role, user navigation, competencies) or a possible target entity changes (e.g. content or access rights of entities in the knowledge are created newly or change). The processing entity 'decides' according to the ECA policy implemented whether or not to propagate the event further.

Fig. 1 gives an overview over the entities of the meta-model and their interrelationships.

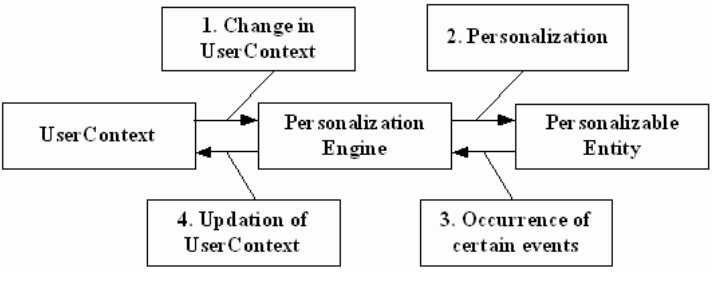



**Fig. 1.** The basic entities of the proposed meta-model

To sum up, dynamic personalization deals with the behavior of a software system according possibly rapid changes in (working) contexts. It provides the user-side as well as the system-side with mechanisms that allow for adapting to these changes and keeps the representation of the system context and the user context up-to-date.

## 2.2 Data and Structure and Context and Behavior

Structural Computing has been established in the last couple of years (see for instance [6]) as a computer science discipline aiming at shifting the attention in software engineering away from the *data* to the *structure*: Instead of viewing the independent data-objects as being of paramount importance, the interrelationships of data objects are viewed as being (at least) equally important. Therefore, instead of analyzing and

designing the essential data objects of a computer system, Structural Computing seeks among others for the basic *structural abstractions* of a system, the *structural transformations* (i.e. transformations from one structures into another) that might be applied to them and for the *behaviors* (i.e. the way a structure acts and reacts upon certain computations) structures might reveal.

In [10] the results of an investigation are described, which aimed at finding the structural abstractions and structural transformations of the meta-model for dynamic personalization introduced above. The structural abstraction needed to provide a way that prevents data objects and connections between them to be viewed and treated separately. They must only be treated as a unity and the structuring abstraction – instead of one distinguished data item – is made the paramount focus of processing at any stage. Fig. 2 gives the class diagram in Unified Modeling Language (UML) notation that represents the structural abstraction.

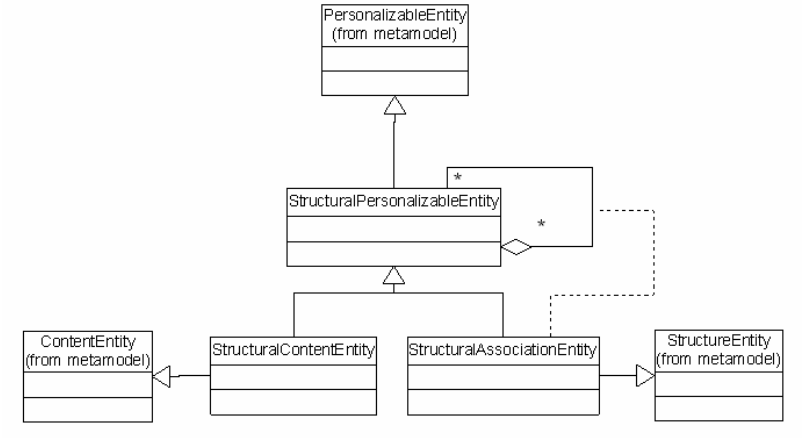



**Fig. 2.** Class structure that allows treating the basic structural abstraction as a unity

The structural transformations described in [10] consist of operations on the structural abstractions that are taken from a taxonomy of Brusilovsky of adaptive hypermedia techniques ([2]). In Fig. 3 an example is given on how the structure is transformed when a new 'element' is inserted into an existing structure.

Although structural abstractions and structural transformations enable software engineers to view a software system from a structure-centric perspective they still have a severe shortcoming: They do not define what actually happens during a structural transformation. They only provide means to ensure that a couple of transformations might take place some time in the future. The inherent dynamics as far as types and forms of different behaviors are concerned are not addressed. The structural abstractions are somehow objects to transformation. The active subject of the transformation is not known. Thus, the semantics of structure are hidden in some other entities such as algorithms and processing instructions, which are not in the scope of the structure.

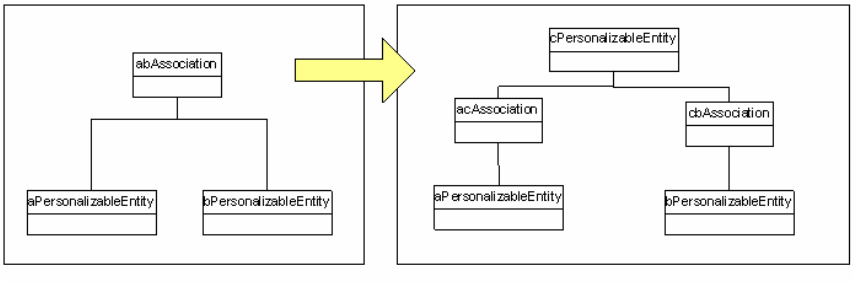



**Fig. 3.** Example for a structural transformation: Inserting a new element

In order to overcome this shortcoming, a couple of investigations have already been made. Stimuli propagation is a way of providing entities with knowledge of how to operate in case a certain input-stimulus affects any other element of the same structural element (Vaitis, Tzagarakis et al. in [11]). In [7] Nürnberg et al. discuss how viewing an entity from a data-view, a structure-view or behavior-view highlights the corresponding data-, structure- or behavior properties of that entity. All three views need to be considered when performing a system-analysis, which aims at fully describing the entities that are computed and the entities that perform the computation.

## 3 The Domain Analysis

In this section we give a brief outline of a project that is currently being carried out in cooperation between the Know-Center, Graz and Graz University of Technology (Austria). The project's objective is to create an educational system which provides lectures, students and alumni with online access to teaching and research material. This section gives an introduction to the domain space in the context of the given project. Then a distinguished feature element, which is referred to as *personal folder*, is described in more detail. Finally, the personal folder is analyzed and its properties are discussed by applying paradigms from Structural Computing.

### 3.1 'Knowledge Transfer' Components in Higher Education

The Graz University of Technology is planning to provide online access to their offerings for lecturers, researchers, students and alumni. Researchers and students will be provided with powerful means for retrieving information from online catalogues and digital libraries. Lecturers will be provided with tools to organize materials, resources as well as with communication and testing features. The project aims at creating an integrated" one stop shop" from which the offerings of the University can be accessed in a convenient way. At a glance, the one stop shop will provide access to the following services (taken from [3]):

- *Research portal*: The portal integrates several heterogeneous and distributed data sources (digital libraries, online catalogues, other research portals), supports relevant query techniques (full-text search, meta-data,

search, similarity search in textual documents as well as multimedia data, fault-tolerant search etc.) and provides automated, autonomous search agents.

- *Lecture library*: Lecturers and tutors will be supported in collecting and organizing recommended readings in their personal library; they make them accessible to peers and learners. Lecture libraries are in essence (small) personal digital libraries which are made accessible just for a given lecture. Lecturers and tutors shall be provided with templates, which allow them to easily set up a lecture web site that looks and behaves in accord with University-wide or Institute-wide requirements. Students as well as peers have access to lecture libraries based on a role-based access rights concept. Search results from the research portal are imported into the personal library and filed under the structuring policy of the personal library.
- *Workspace*: The project creates workspaces for groups of students and individuals. Lecturers, tutors and students can set up workspaces. Workspaces may contain personal digital libraries, lecture libraries, appointment calendars or upload areas. Lecturers (and researchers) can set up workspaces that are devoted to certain lectures or fields of interest and resources from their workspaces can be made accessible via lecture libraries. Resources from workspaces set up by students can not be moved to lecture libraries but can only be re-used in the student's personal digital library.
- *Personal folder:* Lecture libraries as well as workspaces provide users (e.g. students, lecturers) with a structured, customized view to the overall content and system features. The basic structuring mechanisms are folder structures. These folders are adaptable to the specific need of the user.

The system aims at supporting teaching and learning processes of students, lecturers and researchers.

### 3.2 Personal Folders

As mentioned above, the basic underlying structuring mechanisms are personal folders that store entities such as readings, collections of readings, uploaded content or lists of appointments. Investigations of areas like eCommerce show that personal folder-like mechanisms are considered to be highly relevant. [1] reports on observations of business-oriented web sites where the concept of personal home pages is considered to play a critical role in driving utilization of a web-site. Utilization is argued to be the most important metric for defining the return on investment (ROI) of any web site. Personal home pages are areas of web sites that are adapted to a specific user's needs and interest such, that all content, features and self-service applications that are relevant to the user, are only a minimal number of clicks away. Personal folders are somewhat similar in that they aim at providing users with an adaptable collection of resources that best suffice the demands of the user's current context.

There are a number of basic operations on personal folders such as creating folders, applying meta-data to folders (e.g. name, date author), adding content to it, changing content or meta-data or deleting folders. Other operations appear to be more sophisticated, for instance versioning of folders and their content, exporting folders to external objects, applying annotations, indexing folder's contents for later retrieval, creating links to other folders or making folders publicly available.

Within the context of dynamic personalization, there are still more operations, a personal folder should offer. Personal folders need keeping track of their content and usage and they also need to provide means to compare themselves to other entities and adapt themselves according to the results of the comparison.

At a glance, personal folders take care of the following dynamic personalization features:

- User context: The context in which a user is, for instance identity, role, work activities (compare section 2).
- System context: Personal folders also have to consider the state in which the system is. This includes among others the personal folder's contents, the folders of which the personal folder is content of and the specific purpose of the personal folder in the context of the policies set up for the system environment. The observation of the folder's content can be realized by software agents that observe the personal folder at regular intervals and keep track of changes to the folder's state.

User context and system context need to be observed (for instance by software agents). Changes to the user's context, the system's context or the personal folder trigger a number of mutual updating operations. The following scenario will clarify this:

A lecturer has set up a personal workspace for a lecture on 'Knowledge Management'. At a given point in time, one of her personal folders contains a collection of resources that are about 'eLearning', which is one distinguished topic of the lecture.

The user context is defined by the user's identity (say user A), the user's current role as a lecturer, the user's current activity (providing resources for this term's students of Knowledge Management), the user's state as an expert for Knowledge Management and the user's fields of interest, which can be extracted and identified from the resources he is working with.

The system context is as follows: The workspace created is the one stop shop for the lecture. Its aim is to provide students with a one-stop portal containing the relevant resources for the lecture. There are a number of references from student's workspaces to the lecturer's workspace and, since the lecturer is a renowned expert in Knowledge Management, there are also a number of references to the workspace from other researchers and experts.

While doing research, the lecturer finds a resource on eLearning dealing with 'Legitimate Peripheral Perception'. The lecturer considers this resource being highly important and puts it into the personal folder on eLearning. The personal folder realizes the change to its content. It propagates the change to all other folders that are linked to the given folder. Students are provided with the new and updated

information. The relationships between the students' folders and the lecturer's folder have been set up at the beginning of the lecture and the behavior is well-defined during run-time since it has been designed in advance.

Another researcher (say user B) also had established a relationship with the workspace on Knowledge Management in the past. The personal folder linked to the lecturer's personal folder on eLearning receives the notification. User B has already stored the contribution on 'Peripheral Legitimate Participation' in his personal folder in the past. The personal folder of user B informs the personal folder of user A and user A's personal folder establishes a relationship with user B's folder. Thus, a two-way relationship between the resource-spaces of personal folders of two researchers is established. The lecturer's personal folder has dynamically been personalized to his specific needs. When user A logs into the workspace for the next time, he is provided with the newly established link to user B and can decide whether or not to use the new source of resources for his work. The relationship between user B's folder and the lecturer's folder has been established at run-time; the way folders behave has not fully been defined during design time but is subject to dynamic changes and adaptations.

This scenario and the considerations described above yields the following insights: Providing functionality dealing with static properties of folders and interrelationships among them is mainly covered by operations that may fully be described at design-time. With requirements from dynamic personalization, a demand for more 'active' functions of personal folders arises. This means a personal folder is responsible for executing operations which affect the folder itself as well as other folders. A number of behavior-concerned functions need to be provided by the folder and to be executed at run-time in accordance with user contexts and the system context.

Currently, behavior-concerned functionality is only supported to a small degree by software systems. A reason might be the insufficiency of methods for analysis and design methods to cover behavior-concerned aspects. In the next section personal folders are examined with special regard to their behavior.

### 3.2 Applying Paradigms from Structural Computing

We consider a folder to be the entity, which represents the most atomic structural abstraction of the system in question. For the following reasons we believe this is true: A folder is considered to be at least some form of structural abstraction: Firstly, is structured in that it is related with other folders and it is structuring in that it represents relations among other folders and entities (compare [7]). Secondly, there is no way to split up the structuring mechanism provided by the personal folder, i.e. it is not possible to find a more basic entity that represents relations between the system's entities in a sufficient way. The other concepts introduced above such as workspaces, appointment calendars and so forth are built on top of personal folders.

In the following, a discussion on notable properties of entities when seen from a data view, a structure view or a behavior view ([7]) is applied to personal folders. Seen from a structural view, a personal folder is characterized by the following properties. It consist of a unique name within the folder (file) system, a set of properties such as creation date and a set of references (e.g. to other folders).

**Table 1.** Discussion of data-view, structure-view and behavior-view of personal folders

| Feature | Structure view | Data view | Behavior view |
|---|---|---|---|
| **Create folder** | ***Name*: Folder ID (the same for all operations)**<br>**Properties: Creation date, creator etc.**<br>**References: empty** | ***Name*: Folder ID (the same during all operations)**<br>**Content: Creation date, creator etc.**<br>**Properties: empty** | ***Name*: Folder ID (the same during all operations)**<br>***Elements operated*: empty**<br>***Operations*: Propagate creation to system context (parent folder, linking to current folder)** |
| **Add reference** | ***Properties*: Owner, creator of copy, original location etc.**<br>***References*: Reference set plus new reference (pointing)** | ***Content*: Bytes representing reference are added**<br>***Properties*: New reference** | ***Elements operated*: Reference set plus new reference**<br>***Operations*: Propagate add operation to system context (parent folders); add new reference for tracking, indexing etc. operations** |
| **Change reference** | ***Properties*: Last update, updating person etc.**<br>***References*: Changed reference (pointing)** | ***Content*: constant**<br>***Properties*: constant** | ***Elements operated*: constant**<br>***Operations*: Propagate change operation to system context (parent folders); remove old reference; add changed reference for operations** |
| **Delete reference** | ***Properties*: Last update, updating person etc.**<br>***References*: Reference set minus old reference** | ***Content*: Bytes representing old reference are deleted**<br>***Properties*: Remove reference** | ***Elements operated*: Reference set minus old reference**<br>***Operations*: Propagate delete operation to system context (parent folders); remove old reference** |
| **Add content** | ***Properties*: Owner, creator of content etc.**<br>***References*: Reference set plus new reference (including)** | ***Content*: Bytes representing content are added**<br>***Properties*: New content added** | ***Elements operated*: Reference set plus new (included) reference**<br>***Operations*: Propagate add operation to system context; index new content** |
| **Change content** | ***Properties*: Last update etc.**<br>***References*: Changed reference (including)** | ***Content*: Bytes representing content are changed**<br>***Properties*: constant** | ***Elements operated*: constant**<br>***Operations*: Propagate change to system context** |

| | | | |
|---|---|---|---|
| **Delete content** | ***Properties*: Last update etc.**<br>***References*: Reference set minus old reference (including)** | ***Content*: Bytes representing content are deleted**<br>***Properties*: Old content removed** | ***Elements operated*: Reference set minus old (included) reference**<br>***Operations*: Propagate delete operation to system context; remove old content from index** |
| **Add meta-data** | ***Properties*: Owner etc.**<br>***References*: Reference set plus new reference to meta-data (including)** | ***Content*: Bytes representing meta-data added**<br>***Properties*: New meta-data bytes** | ***Elements operated*: Reference to meta-data added**<br>***Operations*: Propagate add operation to system context** |
| **Change meta-data** | ***Properties*: Last update etc.**<br>***References*: Changed reference (including)** | ***Content*: Bytes representing meta-data changed**<br>***Properties*: Changed meta-data bytes** | ***Elements operated*: constant**<br>***Operations*: Propagate change to system context** |
| **Delete meta-data** | ***Properties*: Last update etc.**<br>***References*: Reference set minus old reference (including)** | ***Content*: Bytes representing meta-data deleted**<br>***Properties*: Old meta-data removed** | ***Elements operated*: Reference to meta-data removed**<br>***Operations*: Propagate remove operation to system context** |
| **Add attribute** | ***Properties*: Last update etc.**<br>***References*: Reference set plus new reference** | ***Content*: Bytes representing new attribute**<br>***Properties*: New attribute bytes** | ***Elements operated*: constant**<br>***Operations*: Propagate add operation** |
| **Change attribute** | ***Properties*: Last update etc.**<br>***References*: Reference changed** | ***Content*: Bytes representing changed attribute**<br>***Properties*: constant** | ***Elements* operated: constant**<br>***Operations*: Propagate change to system context** |
| **Delete attribute** | ***Properties*: Last update etc.**<br>***References*: Reference set minus old reference** | ***Content*: Bytes representing attribute deleted**<br>***Properties*: Attribute removed** | ***Elements operated*: Reference to attribute removed**<br>***Operations*: Propagate delete to system context; remove attribute from indexing etc.** |
| **Create Version** | ***Name*: New Folder ID for new version**<br>***Properties*: New version number**<br>***References*: Reference set contained stays the same** | ***Name*: New Folder ID**<br>***Content*: New byte block allocated; content from old version plus new version number copied**<br>***Properties*: Same as old version** | ***Name*: New Folder ID**<br>**Elements operated: Same as old version**<br>***Operations*: Propagate new version to system context; parent folders triggered to add reference to new version to their reference set** |

| | | | |
|---|---|---|---|
| **Roll back version** | ***Name*: Old Folder ID**<br>**Properties: Old version number**<br>***References*: Old version reference set** | ***Content*: Bytes representing old version**<br>***Properties*: Old version's properties** | ***Elements operated*: Old version's elements**<br>***Operations*: Propagate roll-back to system context; parent folders triggered to roll-back reference set to new version** |
| **Print** | ***Properties*: constant**<br>***References*: constant** | ***Content*: constant**<br>***Properties*: constant** | ***Elements operated*: constant**<br>***Operations*: References printed according to print order** |
| **Export** | ***Properties*: constant**<br>***References*: constant** | ***Content*: constant**<br>***Properties*: constant** | ***Elements operated*: constant**<br>***Operations*: References exported according to export order** |
| **Add annotation** | ***Properties*: Last update etc.**<br>***References*: Reference set plus new reference (pointing/ including)** | ***Content*: New bytes representing annotation**<br>***Properties*: New annotation** | ***Elements operated*: Reference set plus new references**<br>***Operations*: Propagate new annotation to system context; add annotation to indexing etc.** |
| **Change annotation** | ***Properties*: Last update**<br>***References*: Reference changed (pointing/ including)** | ***Content*: Bytes representing changed annotation**<br>***Properties*: Changed annotation** | ***Elements operated*: constant**<br>***Operations*: Propagate change to system context** |
| **Delete annotation** | ***Properties*: Last update etc.**<br>***References*: Reference set minus old reference (pointing/ including)** | ***Content*: Bytes representing annotation deleted**<br>***Properties*: Annotation removed** | ***Elements operated*: Reference to annotation removed**<br>***Operations*: Propagate delete to system context; remove annotation from indexing etc.** |
| **Index folder** | ***Properties*: constant**<br>***References*: Reference to index added (hidden)** | ***Content*: Bytes representing new index**<br>***Properties*: New index** | ***Elements operated*: constant**<br>**Operations: New index propagated to system context** |
| **Search in folder** | ***Properties*: constant**<br>***References*: constant** | ***Content*: constant**<br>***Properties*: constant** | ***Elements operated*: constant**<br>***Operations*: Retrieve information from index** |

| | | | |
|---|---|---|---|
| **Create search agent** | ***Properties: constant References: Reference set plus new reference to agent (pointing)*** | ***Content: Bytes representing reference to agent Properties: New search agent*** | ***Elements operated: New reference to agent Operations: Schedule agent; Propagate search results of agent to system context in regular intervals according to search agent's policy*** |
| **Change search agent** | ***Properties: constant References: Changed reference to agent*** | ***Content: Bytes representing agent changed Properties: Changed search agent*** | ***Elements operated: Changed reference to search agent Operations: constant*** |
| **Delete search agent** | ***Properties: constant References: Reference set minus reference to agent (pointing)*** | ***Content: Bytes representing reference to agent changed*** | ***Elements operated: Reference to search agent removed Operations: Propagate to system context*** |
| **Link from other folder created** | ***Properties: constant References: Reference set plus "backward" reference to other folder*** | ***Content: Bytes representing "backward" reference added Properties: New "backward" reference*** | ***Elements operated: Other folder added to system context Operations: Propagate to user context*** |
| **Track user context** | ***Properties: constant References: constant*** | ***Content: Bytes representing user context (possibly) changed Properties: constant*** | ***Elements operated: constant Operations: Trigger event if user context changes; propagate event to system context; folders linked connected with current folders are analysed for results that fit new user context*** |
| **Track system context (state of folders linking to current folder)** | ***Properties: constant References: constant*** | ***Content: Bytes representing system context (possibly) changed Properties: constant*** | ***Elements operated: constant Operations: Trigger event if system context changes; propagate event to user context; Propagate event to folders connected with current folder*** |
| **Track personal folder** | ***Properties: constant References: constant*** | ***Content: constant Properties: constant*** | ***Elements operated: constant Operations: Trigger event in case personal folder changes; Propagate event to system context; propagate event to user context*** |

Seen from a data view a folder is made up of a unique name, an arbitrary content (which might be interpreted by the system as containing semantic information for instance whether the folder is a personal digital library or lecture library) and a set of properties such as access rights.

From a behavior view, a folder consists of a unique name and a set of elements on which functions can operate and a set of operations. Depending on the type of folder (personal folder, lecture library etc.), there are several instructions, which might be executed on the elements of the operation. The interpretation of the semantics, i.e. the type of the folder and what operations might be executed is subject to personal folder operations.

The examination of the data-structure-behavior views of personal folders reveals the following results: Personal folders are examined and described for their properties, interrelationships and behaviors relevant in the context of each view. Through this, especially behavior-concerned aspects are made visible and can more easily be described. When realizing the system, special care is taken for behavior-concerned system requirements and thus, for features that are necessary to realize active dynamic personalization.

The following table provides an overview of the characteristics which need to be considered when viewing personal folders from different perspectives. Each column represents a specific view on personal folders. The table cells show some properties that are highlighted when viewing personal folders from the according view.

## 4 Discussion an Open Issues

The concept of personal folder is considered to be the basic structuring mechanism of our system for 'knowledge transfer' in higher education. Personal folders show several interesting characteristics given a number of specific user contexts and system contexts. Some of the characteristics of personal folders are more related to the structural nature whereas some are more related to their behavioral nature. In this contribution, an examination has been outlined, which aimed at analyzing personal folders from different views (data, structure, behavior). Each of the views only highlights the properties that are considered to be relevant for the respective view. Putting the results together leads to a broader picture of the object of investigation. We therefore believe that such a multidimensional approach during the analysis phase helps to better identify the essential characteristics of a software system compared to an isolated analysis of the application domain from just one viewpoint.

It is certainly too early to know for sure whether or not the properties which have been revealed during the examination are complete and represent static as well as dynamic aspects of a system in its entirety. One reason for this is that completeness is an open issue we identified during our research. It would be a great help if some mechanisms would be available to get a feeling of how "complete" the result of the analysis process is. Also, we are unsure about the impact of an incomplete analysis: For example, how strong is the impact of an incomplete analysis from one view (e.g., data) on the analysis results in another view (e.g. behavior). A suggestion for future basic research would be to better understand the interdependencies between the different views of the grand unified theory.

## 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. Broadvision: Using Proven Personalization Techniques to Drive Measurable and Profitable Online Behavior. Broadvision White Paper, 2004.
2. Brusilovsky, P.: Adaptive Hypermedia. User Modeling and User-Adapted Interaction, Vol. 11, Kluwer Academic Publishers, Dordrecht, Netherlands. (2001). 87-110
3. Graz Digital Library – Functional Specification, Technical Report, Know-Center (2004).
4. Hicks, D. L. (Ed.): International Symposium, MIS 2003 Revised Papers. Lecture Notes in Computer Science , Vol. 3002. Springer-Verlag, Berlin Heidelberg New York (2004).
5. Kandpal, D.: Augmenting knowledge-intensive systems with Dynamic Personalization concepts. Doctoral Thesis, Graz Universitiy of Technology (2003).
6. Nürnberg, P.: Repositioning Structural Computing. Lecture Notes in Computer Science, Vol. 1903. Springer-Verlag, Berlin Heidelberg New York (2000) 179-183
7. Nürnberg, P.J., Wiil, U.K., Hicks, D.L.: A Grand Unified Theory for Structural Computing, Lecture Notes in Computer Science, Vol. 3002. Springer-Verlag, Berlin Heidelberg New York (2004) 1-16
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. Ulbrich, A., Kandpal, D., Tochtermann, K.: Dynamic Personalization in Knowledge-Based Systems from a Structural Viewpoint. Lecture Notes in Computer Science, Vol. 3002. Springer-Verlag, Berlin Heidelberg New York (2004) 126-142.
11. Vaitis, M., Tzagarakis, M., Grivas, K., Chrysochoos, E.: Some Notes on Behavior in Structural Computing. Lecture Notes in Computer Science, Vol. 3002. Springer-Verlag, Berlin Heidelberg New York (2004) 143-149.

# Content Engineering: Bridging the Gap Between Content Creation and Consumption

Siegfried Reich

Salzburg NewMediaLab,
Jakob Haringer Straße 5/III,
5020 Salzburg, Austria
sreich@salzburgresearch.at

**Abstract.** We have ever more opportunities to create content, e.g. using digital cameras, or, taking photos with mobile phones and publishing them on the Web, to name just a few. At the same time, as content consumers, we have a growing need for context-aware, tailored content, e.g. location and time-based services delivered via wireless LAN to our PDA. Thus, as content *creators* we aim at higher levels of re-use and as *consumers* we expect individualised content at high quality.

There is a gap, one may call it a "content crisis" even, between the growing opportunities for content creation and the increasing needs raised by content consumption. Content Engineering as a discipline may be a step forward to bridge that gap.

## 1 Introduction

Many – not to say most – of us are Web users. For instance statistics available at http://www.internetworldstats.com/stats.htm argue with growth rates well above 100% for the last four years throughout the globe resulting in almost 70% of North Americans using the Internet, in Europe it is still more than 30% of the population (in 2004).

Besides the aspect of access and/or consumption, we are also increasingly playing the role of content creators. For instance, in Japan, the penetration rate with mobile phones with built-in cameras is expected to reach 100% in 2005 (see http://www.eurotechnology.com/store/camera-phone): there will be more mobile phones with cameras than there are cameras. Additionally, common infrastructures are emerging, e.g. the MPEG family on a coding level, or SMIL (the synchronised multimedia integration language) on an integration level. Future cameras for instance, will not only be able to render SMIL but will also be able to directly produce SMIL encoded content.

Additionally, we are all working towards "smart" content applications [1], often in the form of location-aware, time-aware and personalised content and one of the issues we are facing is that content is not available in the respective granularity, i.e., we get canned content which is targeted to specific platforms and scenarios and which is therefore not re-usable.

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 206–211, 2004.

Thus, we are confronted with a gap between the amount of content available in general and its usefulness as measured in user acceptance. Some have therefore spoken of a crisis; *Klas* [8] for instance argues that multimedia producers and distributors are facing a formidable management task (a "multimedia engineering crisis" in analogy to the "software crisis" of the sixties and seventies). Some argue more for new challenges, e.g. *Fonnesbech* [5] identifies a need for "dramatic content engineering" whereby he means that only by working *both* effectively *and* creatively on a foundation of solid knowledge about the audience, the business model and the technological platform, will we be able to create successful content, i.e., content that is being accepted by the users.

In the following we argue for Content Engineering as a new discipline that should help in addressing these issues.

## 2 Content Engineering: A Definition

The term engineering in general means "the practical application of science to commerce or industry"[1]. An engineer must identify and understand the relevant constraints in order to produce a successful result. Constraints include available resources, physical or technical limitations, flexibility for future modifications and additions, and other factors such as requirements for cost, manufacturability, and serviceability. By understanding the constraints, engineers deduce specifications for the limits within which an object or system may be produced and operated. The term "engineer" has its origin in the Latin word "ingeniosus" meaning "skilled". Hence, engineers are skilled problem solvers.

In computer science, we are familiar with several types of engineering, for instance software engineering [3], or, more recently, also with disciplines such as Web Engineering [4, 7] or Hypermedia Engineering [2, 9].

So why would we want to apply engineering principles to content? We argue that we need to address re-use as well as tailorability in order to be better able to bridge the gap outlined in the introduction.

- Re-use: Re-use can be defined broadly as "use again after processing"[2]. There are several types of content items that would be subject to re-use: ideas for instance, designs, code, media, processes or also combinations thereof. Secondly, we need to define how to measure the degree of re-use. Examples of metrics could include the number of re-use in different publications, the amount of revenue generated, the amount of time spent for adaptation for an artefact to be re-usable.
  Within the various types of re-use, there are also differences. The degree of automation of code re-use in software engineering is still limited. On the other hand, if we think of media re-use, then we (as human beings) may well

[1] See http://www.webster-dictionary.org/definition/engineering
[2] See http://www.webster-dictionary.org/definition/reuse

be able to read an automatic translation of a German text into English even though the grammar may not be correct: as humans we are typically more flexible in perception and therefore a basic level of re-use may be achieved more easily for those items.
- Tailorability: Tailorability can be defined as tailor-made, i.e., made or as if made specifically for the particular purpose at hand[3]. This again is a huge topic, ranging from formatting issues such as screen real estate, bandwidth, the delivery platform's hardware and software capabilities up to real issues i.e., targeting a content artefact towards a user's location, the respective time, the user's interest, and many more.

There are existing definitions of "Content Engineering". E.g. *Vliet* defines Content Engineering as "the development of information systems that support the entire value chain of multimedia production or parts thereof: creation, digitisation, storage, search, manipulation, management, distribution and delivery, in an effective, efficient and user friendly way" [11]. In our view this is more a definition of engineering content management solutions and it is focused too much on systems only. *Maicher* defines "Content Engineering as the searching, acquisition, classification, storage and visualisation of contents which is supported by processes and tools [10]". This definition is based on a sequential list of activities and misses a conceptual layer.

Following these definitions, we define Content Engineering as "the application of systematic and quantifiable approaches (concepts, methods, techniques, tools) along the content value chain, i.e., content acquisition, value adding, and distribution and delivery, in order to support re-use and tailorability for media-rich publications."

With this definition we get the content value chain as a top level concept with defined phases covering the whole life cycle of media artefacts (see Figure 1). Also, we argue what we mean by content ("media rich artefacts") and we present an idea of which approaches we may use, e.g. a formal language (concepts), a systematic study (methods), genres (methods), best practice (techniques) and industrial production (tools).

With respect to the artefacts themselves, we believe that a separation of media, indexing structures, design, etc. is key to ensure re-use. *Wurman* [12] for instance argues for the ways of organising information to be finite, in particular he argues for the dimensions location, alphabet, time, category and hierarchy (LATCH) to be the core dimensions of information organisation and that content should be organised along these dimensions (see also Figure 2).

A different way of viewing this is to think of autonomous content objects which are being passed through the value chain and offer different interfaces to the various publishing activities. See for instance enhanced multimedia meta objects (EMMOs [6]).

Finally, with respect to the process models involved in the various activities we believe that different approaches would be possible. Just as there are

[3] http://www.webster-dictionary.org/definition/Tailor-made

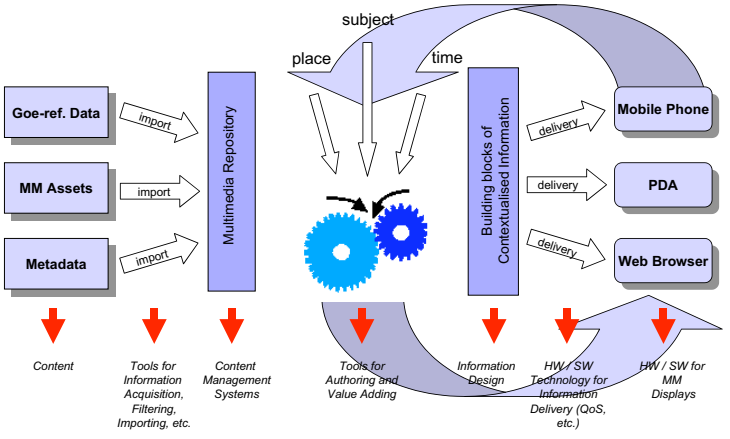



**Fig. 1.** Content Value Chain

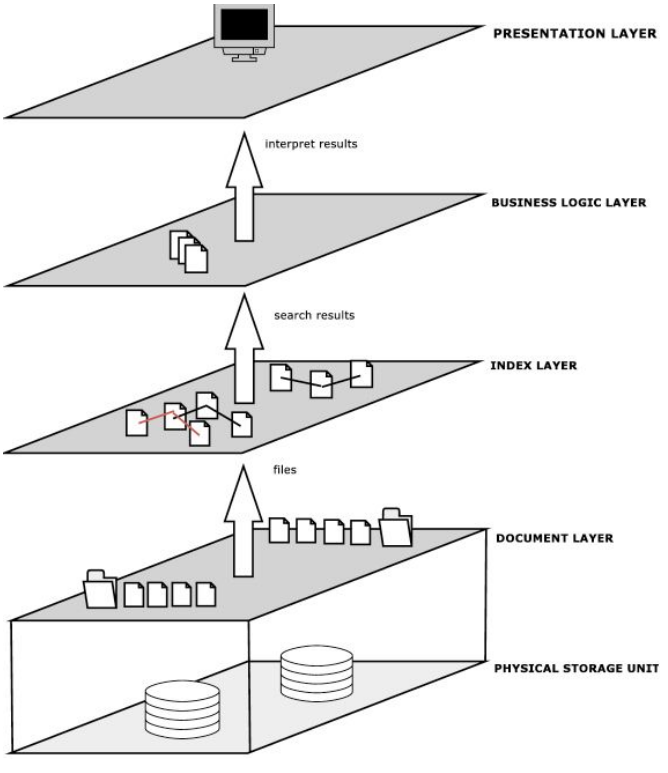



**Fig. 2.** LATCH Layers

agile methods and more traditional methods (e.g. waterfall model) in software engineering, we argue that different styles — or genres — of content creation and manipulation need to be defined. For instance engaging in an online chat is different to writing a thesis; a weblog would have different requirements as well. The information architecture using e.g. LATCH layers as outlined in Figure 2 would be a basic concept for enabling these different types of narratives being put on top of existing media artefacts.

## 3 Summary

In this position statement we have argued for a gap we perceive in the possibilities of creating content and the expectations we have in consuming situation-aware, tailored content. We believe that "content engineering" as a discipline would provide a structured approach which would help in addressing these issues by defining a content value chain, by outlining the necessary (logical) architectures and by allowing different process models to be included.

## Acknowledgements

We would like to acknowledge the support of Salzburg NewMediaLab via the Competence Center Programme funded by the Austrian Ministry of Economic Affairs and Land Salzburg under grant No. 98.362/65-I/16/03.

## References

1. Wernher Behrendt, Guntram Geser, and Andrea Mulrenin. EP2010 — the Future of Electronic Publishing Towards 2010. Information Society DG – Unit E2, EUFO 1-275, Rue Alcide de Gasperi, L-2920 Luxembourg, 2003.
2. Michael Bieber. Hypertext and web engineering. In *Proceedings of the '98 ACM Conference on Hypertext, June 20-24, 1998, Pittsburgh, PA*, pages 277–278. ACM Press, 1998.
3. Barry W. Boehm. Software engineering. *IEEE Transactions*, 25(12):1226–1241, 1976.
4. Yogesh Deshpande, San Murugesan, Athula Ginige, Steve Hansen, Daniel Schwabe, Martin Gädke, and Bebo White. Web engineering. *Journal of Web Engineering*, 1(1):3–17, 2002.
5. Christian Fonnesbech. Dramatic content engineering. The ACTeN Project, 2002. `http://www.acten.net/cgi-bin/WebGUI/www/index.pl/newsletter1?wid=183&func=viewSubmission&sid=106`.
6. Sunil Goyal, Wernher Behrendt, and Siegfried Reich. EMMOs — enhanced multimedia meta objects for re-purposing of knowledge assets. In *MIS '02 - Meta Informatics Symposium 2002, Esbjerg, August 2002, LNCS 2641*, pages 155–160, 2002.

7. Gerti Kappel, Birgit Pröll, Siegfried Reich, and Werner Retschitzegger, editors. *Web Engineering — Systematische Entwicklung von Web-Anwendungen.* dpunkt Verlag, Heidelberg, 2003. In German.
8. Wolfgang Klas. Describing multimedia. Technical report, Multimedia Information Systems, University of Vienna, September 2000. `http://www.cordis.lu/ist/ka3/iaf/swt_presentations/swwsklas.htm`.
9. David Lowe and Wendy Hall. *Hypermedia & the Web. An Engineering Approach.* John Wiley & Sons, Chichester, 1999.
10. Lutz Maicher. On how to model content engineering in a semantic web environment. In *Innovative Internet Community Systems, Third International Workshop, IICS 2003, Leipzig, Germany, June 19-21, 2003, Revised Papers, LNCS 2877*, pages 168–179, 2003.
11. Harry van Vliet. Directions for long-term content engineering research. A view from above. Technical Report TI/RS/2000/048, Telematica Instituut, the Netherlands, 7500 AN Enschede, 2000. `https://doc.telin.nl/dscgi/ds.py/Get/File-9796/GigaCE_Long_term_Content_Engineering_Research.pdf`.
12. Richard Saul Wurman, David Sume, Loring Liefer, and Karen Whitehouse. *Information Anxiety 2.* Que, December 2000.

# Blog Perspectives

## Services: Amoeba Versus Whale

Frank Wagner

Roskilde University,
Universitetsvej 1, 4000 Roskilde, Denmark
frankw@ruc.dk

**Abstract.** Information technology is supposed to provide services and to support work. The term service is used in different ways, for both parts of a system and for what a company provides in electronic form. Common is that it describes the encapsulation of some functionality so it can be used to process data in a well defined way.

The use of information technology is changing though. It is now supposed to serve individual users with their knowledge work and it is supposed to improve cooperation between organisations and access in general. Applications as they are used now leave too much work to the user and even require extra work around their limitations and to learn to use them.

Structural computing tries to provide a new kind of services working with structures instead of data to support knowledge work. Web services on the other side are an attempt to enable cooperation between organisations. Both have a problem in relation to individual users.

At the same time new applications like blogs and wikis gain ground. They adopt more easily to their users needs and are successfully used both in private and for knowledge work. They express a different view on the use of information technology. This paper tries to show perspectives blogs and similar applications can give on services and system architecture.

## 1 Services: Amoeba Versus Whale

Amoeba versus whale - how does this make sense in the context of this paper? It depends on the perspective. Both are encapsulated by some layer that controls interaction with their surroundings, but neither has a layered achitecture. Even though their size does not matter much when looking at the overall system architecture, it does matter when we as humans try to interact with them. This is not just a question about the interface, but it has to do with the architecture. Well, we rarely interact with amoebas or whales directly, but we can not all really avoid interacting with information technology.

In section 2 I describe some of the problems related to applications and how they may be related to an attempted layered system architecture. Section 3 is about blogs, a kind of applications that seem to emerge from the needs of their users. Section 4 is about perspectives and I want to focus about some aspects found when looking at blogs and that I consider relevant for talking about

U. Kock Wiil (Ed.): MIS 2004, LNCS 3511, pp. 212–219, 2004.

applications. In section 5 I try to describe a simple service oriented architecture, where applications make use of the services information technology can provide instead of being applications provided by the technology.

## 2 Practical Problems with a Layered Architecture

Applications are considered to be part of a layered system archtecture, where they are in the top layer, which interfaces to the user. Even though system development pays much attention to the human-computer interaction and interface design has been improved, these applications do not seem to be as useful as expected, especially for knowledge work.

A layered system architecture has been good practice for some time now, each layer encapsulating the implementation details and providing an interface to the layers beneath and above. The lowest layer with an interface to the hardware and the highest layer, the application layer, with an interface to the users. The whole system is used as a tool. The interaction happens through the outer interface of the application or application layer.

Applications developed with this layered approach in mind have been useful for formal organisations, where the needs can be welldefined and the applications can be adopted to them. The use of layers makes it easier to move and reuse pieces in different systems. For informal or individual use, these applications and systems seem to be less useful and they require quite some effort from the users. There has been focus on the user interface to improve the usability of the applications, but this does not solve all the problems. There are both practical and conceptual reasons for these problems.

To meet different individual and more spontanous needs, as they are known from knowledge work, the applications have been extendend. This has resulted in, compared to specific individual needs, sometimes huge applications. Specialised applications are still needed. These applications overlap in the application layer and can interfere with each other. A layered architecture requires some control which is not suitable for individual use.

Applications seem to need access to lower layers of the system and they often spread themselves all over the system. The system may try to control at least the integrity of the system and to prevent applications to access the lower layers, but this may invalidate the application if it depends on it. The system may protect the running applications from each other, but it can not protect the data these applications work on. Furthermore user specific data, both about the configuration and the content itself is spread all over the system.

The application logic itself expresses the developers' understanding of the users' (generalised) needs. It relies on the underlying layers and logic, so the adoption to individual use often requires knowledge of those. An example for such a mixed description can be found in a description of a blog system [6].

Another reason may be found in the nature of knowledge work itself [2]. Knowledge can be used to evaluate a variety of possible solutions and to decide what to do. What actually is done depends on the situation. An organisation

often has defined rather fixed processes, so organisational work may be seen as a subset of knowledge work in general.

While a layered system architecture seems to be useful for formal organisations, it does not support applications built to meet the needs of both individual users and knowledge workers. Applications that adopt to individual use without being too expensive in neither training nor professional assistance may need a different architecture. This may be useful for organisations as well, because individual have to act together in the context of the organisation and because some organisational work has to be coordinated across organisations [8]. Apart from that, large development projects resulting in large applications seem to be very difficult to control.

The system architecture should lead to a clear separation of concerns. The nature of knowledge work requires support by applications that do not fit into the architecture currently used for personal systems. The problems are not as much about the data and the funcionality as they are about how these are used and about the resulting complexity of the applications.

## 3 Blogs and Co

The term blog (or weblog) is used for both the way information is provided and for the tools used to do it. Blogs are used by many different people and they are used in many different ways. The following quote summarises the characteristics I want to focus on.

> Weblogs are important new components of the Internet. They provide individual users with an easy way to publish online and others to comment on these views. Furthermore, there is a suite of secondary applications that allow weblogs to be linked, searched, and navigated. Although originally intended for individual use, in practice weblogs increasingly appear to facilitate distributed conversations. This could have important implications for the use of this technology as a medium for collaboration. Given the special characteristics of weblogs and their supporting applications, they may be well suited for a range of conversational purposes that require different forms of argumentation. ... [3]

There are several approaches to realise blogs [10]. Livejournal is serverbased; the content is stored on a central server, but there are clients to make it easier to update the journal (blog). Radio Userland is clientbased; the content is on the local system but can be uploaded to a server. Blojsom [14] publishes files in a directory hierachy; it acts on content external to the software, but can be used to maintain it through a web interface or a remote client. Most provide some basic functionality which can be extendend.

I see blogs as an example for a kind of applications that support conversations in different ways. Other examples for this kind of applications are wikis, chats,

email and news. They all work on text but support other kind of information to some degree. The technical differences between these applications are not really big. It is the intention for using them that makes the difference.

The essence of a blog is to write down some impression or thought, without having to think about anything else; just write and save; just like some notes on a piece of paper. Blogs use time as an identifier and this can used to the context. Blogs use to provide some kind of notification about new entries (news- or rss-feeds), that can be used to aggregate changes in several blogs. A blog does not have to be accessible to everyone. After all, it is a first impression and references to entries should stay valid.

Wikis are not organised by time, but many provide change notifications as they are common for blogs. Wikis relate entries (called pages) through references embedded in the text, so while blogs use time to identify entries, wikis use the context. Both are often extended by each other's feature [4]. Email is often used to notify about changes. Email and news support aynchronous more or less private dialogues. Chats support synchronous dialogs.

Blog conversations, mentioned in the quote above, require additional support to be used efficiently. One problem to be addressed is about how they can be discovered and tracked. This is done by using additional information like the referrer and trackbacks [11] to blog entries. It is not very reliable and usually not available for chats, emails and wikis which complement the conversations. Another problem is related to the different perspectives that influence the contributions to such a conversation. There should be support for work with the information about the perspective as it can be essential for the use and understanding; more about this in section 4.

The basic idea of blogs is to somehow capture and express impressions and thoughts as they arise. To begin with this information is related to the context, which includes the perspective in use, by a time reference. While working with this information an as experience grows, additional support by technology is needed. This can be satisfied by services provided by the local or by remote systems. Blogs are closely related to similar applications like wikis, email, news, chat and more and it may be useful to see all of them as the application of the technology depending on and emerging from individual needs in a social context. Lilia Efimovas blog [1] covers many aspects of blogs, f.ex. [2, 4].

## 4 Perspectives

Perspectives and how we use and deal with them are interesting for both blog conversations and application development. Below I want to focus on some aspects of blog conversations and try to find inspiration for application development.

Blogs entries, even in the same blog, can be written using different perspectives. Blogs conversations and blogs in general are not intended to harmonise

these perspectives, instead they may be used in a dialectic or a complementary sense aiming at an improvement of personal or shared concepts. There are several ways to use the information from blogs and there are several problems related to it. I want to use the idea of the concept to address some of these problems.

Concepts are often identified by terms. The presentation of [9] gave an example that a simple mapping of terms in different languages can be insufficient to understand the related concepts. Instead of identifying concepts by terms, concepts can be identified or specified in many different ways. [5] describes an approach using automated text analysis.

> The main difference between our research and the work on ontologies and the Semantic Web is that the latter tries to formalise structures that are believed to exist in some abstract sense in the real world, thereby making classification and inferencing possible (e.g. that apples are fruit). On the other hand, we have developed techniques that pick up the patterns people leave in their weblog which we believe are a result of their use of an underlying conceptualisation. ... [5]

I understand concepts in a way that is similar to 'underlying conceptualisation'.

A concept is the collection of all the impressions and thoughts about some object. These can to some degree be expressed as information of some kind that we can work with and that can be shared. This information can be about certain aspects of the object and may be related to certain perspectives. Impressions, thoughts, pieces of information and more contribute to the concepts and they can be used to verify, validate and otherwise work with each other.

From this perspective, knowledge work is about concepts, not about information. Knowledge work requires skills to handle information so it can be used to enhance the concepts of interest. This approach separates the interpretation and use of information (the knowledge work) from mere manipulation of information. The concept determines how information has to be handled and we have to know how information has been handled to be able to use it for a concept.

The meaning of information is to improve concepts. Information makes sense (it can be used by our senses) if we can use it to improve concepts; information can make sense even if we do not understand or know the meaning and intention behind it. To make this work we analyse, characterise and evaluate the information; this is what we have to learn and what we have to have experience with. Information can be exchanged and shared in an attempt to build shared concepts. Shared concepts can work if the individuals can develop a common understanding. We use trust to make this manageable.

Application development involves usage perspective and system perspective, both being in general very different. The corresponding concepts are very different. Applications as part of the system only leave the interface to mediate between these different concepts. In cases where the usage and the application is welldefined, this may be enough, but it is not in general. Using the idea of concepts described in this section and what I summarised in the section about blogs, applications can be seen as a kind of shared concept between the user of information technology and information technology systems.

## 5 Applications

The term knowledge work is used for a kind of work that may seem to be more creative compared to organisational work which is meant to be more productive; finding what is right to do in contrast to doing it how it is done right. With organisational work being about the organisation itself instead of the production this devision does makes less sense.

This corresponds to the change of terms from data processing to information technology. Not taking account of the point of view, the discussion of the difference between data and information can be end- and fruitless. Information can be processed like data, like a laptop can be used as a paperweight. The question is about finding the right perspective to see the subtle differences that may have important consequences.

Data are defined so they can be processed independently of their meaning; this can be supported efficiently by technology. Information is choosen with an intention and/or to express a meaning which is dependent of the context and perspective; processing information as raw data does not make sense in general. Information is part of some structure which may be rather complex. One way to deal with this complexity is to find patterns we can process.

From this point of view information technology should provide services to improve work with patterns. Structural computing adds additional data types which define and complement other data types. Structure services are examples for how to process these data and can be used to build services to handle patterns.

The concept presented in 4 describes a generic pattern. A service provider should be able to deal with such generic concepts. It encapsulates the system and provides at least one service to accept requests, perform generic security checks and respond to the request. Without any other services supporting it, all it can do is to somehow confirm the request. It may use services from the same provider or from another service provider eventually encapsulating the system even more. Services by the same provider are more trustworthy than services by other providers.

Access to time stamps and to storage of the representations of the request should be enough to provide a (request-) log, but a blog uses the content in several ways. This is where the service architecture should become visible. Instead of defining functionality for the blog inside the service, the service has to be able to recognise the kind of request and to verify as much as possible. It collects at least references to what it can identify and builds its own concept of this request. It then tries to find one or more services that match patterns in this concept.

The idea is to verify as much as possible, but otherwise only do as little as necessary to delegate the request (with its own internal concept) to a more qualified service. The behaviour of the service in case of ambiguity could be influenced in different ways, including preferences and the state and capabilities of the system. If it deviates from the expected behaviour this should be reasonable. The modification of the basic behaviour define the application and should follow the user from provider to provider. How this can be done has to be studied.

The term 'service oriented architecture' is often used with 'web services' in a business oriented context. Web services are not intended to be only used in this context, but it seems as if it dominates the development. An indication for this may be that they often are understood as providers of data and functionality and that the focus is on welldefine business processes.

Web services are in a way similar to whales: highly specialised and we expect some kind of intelligence behind but we don't know really what they are up to. Structure services on the other hand are a bit like amoebas: they can be everywhere, they are mobile and very flexible, but they are still a bit difficult to interact with.

What we may need are services that help us to describe and define patterns and that can be trained so they behave as we need it. The emerging applications should be mobile and flexible.

A service oriented architecture based on concepts and patterns may support individual and social needs better than an architecture based on data or information and functionality. It supports the verification and use of information in the context of the concept. The emerging applications should be as small as possibel and as big as needed. They belong to the user in a context and help to represent the user's concepts. These applications or parts of them should be easier to move between different devices and could even run simultaneosly to both protect the content by distributing it and to utilise the different capabilities of the different devices.

## References

1. Lilia Efimova's web log. *Mathemagenic: learning and KM insights*, http://blog.mathemagenic.com/
2. Efimova, Lilia Thoughts on task-based view of knowledge work *Mathemagenic: learning and KM insights, 2004-07-26*, http://blog.mathemagenic.com/2004/07/26
3. de Moor, Aldo and Efimova, Lilia 2004. An argumentation analysis of weblog conversations. *Proceedings of the 9th International Working Conference on the Language-Action Perspective on Communication Modelling (LAP 2004)*, Rutgers University, The State University of New Jersey, New Brunswick, NJ, USA, June 2-3, 2004, http://blog.mathemagenic.com/categories/phdNews/2004/05/13.html#a1204
4. Efimova, Lilia My dream wiki/weblog tool. *Mathemagenic: learning and KM insights*, http://blog.mathemagenic.com/2004/06/08.html#a1233
5. Anjewierden Anjo, Brussee, Rogier and Efimova, Lilia Shared conceptualisations in weblogs. *Mathemagenic: learning and KM insights*, http://blog.mathemagenic.com/2004/09/06.html
6. van Kesteren, Anne The perfect weblog system *Anne's Weblog about Markup & Style*, http://annevankesteren.nl/archives/2004/08/weblog-system
7. Wiil, Uffe K., Hicks, David L. and Nürnberg, Peter J. An Agenda for Structural Computing Research *Proceedings of Metainformatics Symposium 2004, Lecture Notes in Computer Science*, 15-18 September 2004, Salzburg, Austria Springer-Verlag, Berlin

8. Rubart, Jessica and Richter, Helge Flexible Notifications and Task Models for Cooperative Work Management *Proceedings of Metainformatics Symposium 2004, Lecture Notes in Computer Science*, 15-18 September 2004, Salzburg, Austria Springer-Verlag, Berlin
9. Krestova, Svetlana and Nürnberg, Peter J. Multilingual Dictionary of Lexicography *Proceedings of Metainformatics Symposium 2004, Lecture Notes in Computer Science*, 15-18 September 2004, Salzburg, Austria Springer-Verlag, Berlin
10. A short, incomplete list of blog software and services Blogger (service):
http://www.blogger.com
Blojsom (software):
http://wiki.blojsom.com/wiki/display/blojsom/About+blojsom
Livejournal (service):
http://www.livejournal.com
Movable Type, TypePad (software, service):
http://www.movabletype.org/ Radio Userland (software):
http://radio.userland.com/
Wordpress (software):
http://wordpress.org/
11. Movable Type *A Beginner's Guide to TrackBack*
http://www.movabletype.org/trackback/beginners/
12. Dervin, Brenda *Sense Making*
http://communication.sbs.ohio-state.edu/sense-making/default.html
13. My own blog to get started. *Takeoff*,
http://flyingdog.ruc.dk/blog/
14. About the motivation to use Blojsom *Takeoff*,
http://flyingdog.ruc.dk/blog/takeoff/?permalink=blojsom.txt

# Author Index