WS15. Workshop on Knowledge-Based Object-Oriented Software Engineering (KBOOSE)

Maja D'Hondt[1], Kim Mens[2], and Ellen Van Paesschen[3]

[1] System and Software Engineering Lab
Vrije Universiteit Brussel
mjdhondt@vub.ac.be - http://ssel.vub.ac.be/members/maja/
[2] Department of Computing Science and Engineering
Université catholique de Louvain
kim.mens@info.ucl.ac.be - http://www.info.ucl.ac.be/people/cvmens.html
[3] Programming Technology Lab
Vrije Universiteit Brussel
evpaessc@vub.ac.be - http://prog.vub.ac.be/

Abstract. The complexity of software domains – such as the financial industry, television and radio broadcasting, hospital management and rental business – is steadily increasing and knowledge management of businesses is becoming more important with the demand for capturing business processes. On the other hand the volatility of software development expertise needs to be reduced. These are symptoms of a very significant tendency towards making knowledge of different kinds explicit: knowledge about the domain or the business, knowledge about developing software, and even meta-knowledge about these kinds of knowledge. Examples of approaches that are directly related to this tendency or could contribute to it are knowledge engineering, ontologies, conceptual modeling, domain analysis and domain engineering, business rules, workflow management and research presented at conferences and journals on Software Engineering and Knowledge Engineering and Automated Software Engineering, formerly known as Knowledge-Based Software Engineering. Whereas this community already contributed for years to research in knowledge engineering applied to software engineering but also vice versa, this workshop intended to focus on approaches for using explicit knowledge in various ways and in any of the tasks involved in object-oriented programming and software engineering. Another goal of this workshop is to bridge the gap between the aforementioned community and the ECOOP community .

On the one hand, this workshop was a platform for researchers interested in the symbiosis of knowledge-based or related methods and technologies with object-oriented programming or software development. On the other hand it welcomed practitioners confronted with the problems in developing knowledge-intensive software and their approach to tackling them.

The workshop's URL is http://ssel.vub.ac.be/kboose/.

1 Introduction

Knowledge in software applications is becoming more significant because the domains of many software applications are inherently knowledge-intensive and this knowledge is often not explicitly dealt with in software development. This impedes maintenance and reuse. Moreover, it is generally known that developing software requires expertise and experience, which are currently also implicit and could be made more tangible and reusable using knowledge-based or related techniques. These are general examples of how using explicit knowledge in a multitude of ways and in all phases of software development can be advantageous.

Since object-orientation is derived from frames in frame-based knowledge representations in Artificial Intelligence, object-oriented software development in se has certain advantages for making knowledge explicit. A conceptual class diagram, for example, models domain knowledge. Also, object-oriented programs can be designed in such a way that certain knowledge is represented explicitly and more or less separated from other implementation issues. However, knowledge might require a more declarative representation such as constraints or rules, thus requiring to augment object-oriented software development with these representations. Examples are UML's Object Constraint Language, or recent trends in using business rules and ontologies.

This workshop is a contribution to ECOOP because knowledge-based or related approaches towards object-oriented programming and software development are under-represented in this conference series. Object-orientation facilitates a more conceptual way of capturing information or knowledge, but current object-oriented software development methodologies do not take advantage of this property.

2 Workshop Goals and Topics

With this workshop we tried to provide a platform for researchers and practitioners interested in knowledge-based or related approaches for object-oriented programming or software development, to pursue and encourage research and applications in this area, and to bridge the gap between the ECOOP community and the long-standing community active in the conferences on Software Engineering and Knowledge Engineering (SEKE) and Automated Software Engineering (ASE), formerly known as Knowledge-Based Software Engineering. More specifically, this workshop wanted to address among others the following:

- identify and characterise
  - object-oriented engineering tasks that can benefit from explicitly used knowledge
  - (knowledge-based) approaches that can support object-oriented engineering tasks
  - kinds of knowledge that are useful to make explicit
  - how explicit knowledge can be used

- look for a common life cycle which describes both the conventional software construction and the knowledge-based software construction
- the symbiosis between knowledge-based or related approaches and object-oriented programming
- industrial applications of explicit knowledge in object-oriented software

Topics of interest include, but are by no means limited to:

- software engineering tasks where explicit knowledge can be useful
  - requirements engineering
  - analysis
  - design
  - programme understanding
  - reengineering
  - reverse engineering
  - software evolution
  - software reuse
  - ...
- approaches for making knowledge explicit
  - knowledge engineering
  - ontologies
  - conceptual modeling
  - domain analysis and domain engineering
  - business rules
  - workflow management
  - ...
- how explicit knowledge can be used
  - modeling
  - enforcing
  - checking and verifying
  - ...

3 Workshop Agenda

The major part of the workshop consisted of group work, since we wanted to avoid a conference-like workshop. Hence, each participant was granted a 1-slide or 10-minute introduction to his or her work in relation to the workshop topics. These introductions were followed by a general discussion about knowledge-based object-oriented software engineering, where the topics proposed by the organisers and the participants in pre-workshop e-mail discussions were considered. These topics converged almost naturally to two main topics. In the afternoon the workshop participants split up in two groups, one for each of the main topics. One hour before the end of the workshop, a representative of each group presented the results of the group discussions in a plenary session. Finally there was a workshop wrap-up session.

While this section describes the workshop agenda, it also provides an accurate overview of the structure of this report: summaries of the position papers (Sec. 4), discussion topics (Sec. 5), and an account of the discussions themselves (Sec. 6). A list of the participants is provided at the end.

4 Summaries of the Position Papers

4.1 *Facilitating Software Maintenance and Reuse Activities with a Concept-Oriented Approach*, Dirk Deridder

A major activity in software development is to obtain knowledge and insights about the application domain. Even though this is supported by a wide range of tools and techniques, a lot of it remains implicit and most often resides in the heads of the different people concerned. Examples of such implicit knowledge are amongst others the links between the different artefacts, and the knowledge that is regarded common sense.

In our approach we will store this knowledge in an ontology, which is consequently used as a driving medium to support reuse and maintenance activities. For this purpose the different concepts, that are represented in the artefacts, will be defined and stored in this ontology. Subsequently these concepts will be 'glued' to the different (elements of) artefacts through extensional and intensional concept definitions. This will enable a bi-directional navigation between the concepts represented in the conceptual artefacts and their concrete realizations in the code which is needed for the intended maintenance and reuse support.

An extended version of this position paper was accepted for presentation and publication at the 5th Joint Conference on Knowledge-Based Software Engineering (JCKBSE2002), in Maribor, Slovenia. [3]

4.2 *Domain Knowledge as an Aspect in Object-Oriented Software Applications*, Maja D'Hondt, and María Agustina Cibrán

The complexity of software domains is steadily increasing and knowledge management of businesses is becoming more important. The real-world domains of many software applications, such as e-commerce, the financial industry, television and radio broadcasting, hospital management and rental business, are inherently knowledge-intensive. Current software engineering practices result in software applications that contain implicit *domain knowledge* tangled with the *implementation strategy*. An implementation strategy might result in a distributed or real-time application, or in an application with a visual user interface or a database, or a combination of above. Domain knowledge consists of a conceptual model containing *concepts* and *relations* between the concepts. It also contains *constraints* on the concepts and the relations, and *rules* that state how to infer or "calculate" new concepts and relations [15]. There is a strong analogy between the rules and constraints on the one hand, and *Business Rules* on the other. Business Rules are defined on a *Business Model*, analogous to the conceptual model of the domain knowledge.

A first problem is that real-world domains are subject to change and businesses have to cope with these changes in order to stay competitive. Therefore, it should be possible to identify and locate the software's domain knowledge easily and adapt it accordingly while at the same time avoiding propagation of

the adaptations to the implementation strategy. Similarly, due to rapidly evolving technologies, we should be able to update or replace the implementation strategy in a controlled and well-localized way. A second problem is that the development of software where domain knowledge and implementation strategy are tangled is a very complex task: the software developer, who is typically a technology expert but not a domain expert, has to concentrate on two aspects of the software at the same time and manually compose them. This violates the principle of *separation of concerns* [4] [14] [5], which states that the implementation strategy should be separated from other concerns or aspects such as domain knowledge. In short, the tangling of domain knowledge and implementation strategy makes understanding, maintaining, adapting, reusing and evolving the software difficult, time-consuming, error-prone, and therefore expensive.

4.3 *Supporting Object-Oriented Software Development with Intentional Source-Code Views*, Kim Mens, Tom Mens, and Michel Wermelinger

Developing, maintaining and understanding large software systems is hard. One underlying cause is that existing modularization mechanisms are inadequate to handle cross-cutting concerns. Intentional source-code views are an intuitive and lightweight means of modelling such concerns. They increase our ability to understand, modularize and browse the source code of a software system by grouping together source-code entities that address a same concern. Alternative descriptions of the same intentional view can be provided and checked for consistency. Relations among intentional views can be declared, verified and enforced.

Our model of intentional views is implemented using a logic meta-programming approach. This allows us to specify a kind of knowledge developers have about object-oriented source code that is usually not captured by traditional program documentation mechanisms. It also facilitates software evolution by providing the ability to verify automatically the consistency of views and detect invalidation of important intentional relationships among views when the source code is modified.

An extended version of this position paper was accepted for presentation and publication at the Software Engineering and Knowledge Engineering conference (SEKE 2002) in Ischia, Italy [8].

4.4 *Management and Verification of the Consistency among UML models*, Atsushi Ohnishi

UML models should be consistent with each other. For example, if a message exists between class A and class B in a sequence chart, then a corresponding association between class A and class B must exist in a class diagram. We provide 38 rules for the consistency between UML models as knowledge base. We can detect the inconsistency between models with this knowledge base. We have developed a prototype system and applied this system to a couple of specifications written in UML[13].

4.5 *Making Domain Knowledge Explicit using $\mathcal{SHIQ}$ in Object-Oriented Software Development*, Ragnhild Vanderstraeten

To be able to develop a software application, the software developer must have a thorough knowledge of the application domain. A domain is *some area of interest and can be hierarchically structured* [15]. Domain knowledge *specifies domain-specific knowledge and information types that we want to talk about in an application* [15].

Nowadays, the de-facto modeling language used in object-oriented development is the Unified Modeling Language (UML) [12]. A lot of domain knowledge is implicitly and explicitly present in the models used throughout the Software Development Life Cycle (SDLC) and in the different UML diagrams, or is only present in the head of the software developers, or is even lost through the SDLC. To make this domain knowledge explicit, we will use one language. Our goal is to translate the different diagrams and write the additional domain knowledge down in this language. We propose to use the Description Logic $\mathcal{SHIQ}$ and extensions to make this knowledge explicit and to support the software modeler in using this knowledge. These logics have quite powerful reasoning mechanisms which can be used to check the consistency of the corresponding class and state diagrams.

In this paper we give the translation of state diagrams and constraints written in the Object Constraint Language (OCL) [6] and two examples of implicit knowledge which is made explicit. In the first example, the consistency is checked between a class diagram and a state diagram. In the second example, a change is made to a first design model, this change makes some domain knowledge implicit. By expressing this knowledge in the logic, the reasoning mechanism of this logic can notify the designer if one of the "implicit" rules is violated.

Our final goal is to build an intelligent modeling tool which enables to make implicit knowledge explicit and to use this knowledge to support the designer in designing software applications. The advantages of such a tool is that reuse and adaptability of software is improved and the understandability of the software designs increases.

4.6 *Declarative Metaprogramming to Support Reuse in Vertical Markets*, Ellen Van Paesschen

The implicit nature of the natural relationship between domain models (and the corresponding delta-analyses) and framework instances in vertical markets, causes a problematic hand-crafted way of developing and reusing frameworks. Constructing a bidirectional, concrete, active link between domain models and framework code based on a new instance of declarative metaprogramming (DMP) can significantly improve this situation.

The new DMP instance supports a symbiosis between a prototype-based, frame-based knowledge representation language and an object-oriented programming language. This implies that framework code in the object-oriented language co-evolves with the corresponding knowledge representation. Representing

domain-dependent concepts in the same knowledge representation allows us to transform delta-analyses into framework reuse plans, and to translate changes in the framework code into domain knowledge adaptations, at a prototype-based and frame-based level.

At the level of the domain-dependent concepts the separation of five kinds of knowledge will be represented by KRS', a dialect of the frame-based prototype-based knowledge representation language KRS. To represent frameworks the new instance of DMP provides a symbiosis between KRS' and the object-oriented programming language Smalltalk. By adding an intelligent component to this instance at the level of KRS', domain-dependent concepts and framework implementations can be coupled, making it possible to automatically translate framework adaptations to the domain-dependent level, and to automatically plan reuse of the framework based explicitly on the delta's at the KRS' level.

4.7 *An Agent-Based Approach to Knowledge Management for Software Maintenance*, Aurora Vizcaíno, Jesús Favela, and Mario Piattini

Nowadays, organisations consider knowledge, or intellectual capital, to be as important as tangible capital, which enables them to grow, survive and become more competitive. For this reason, organisations are currently considering innovative techniques and methods to manage their knowledge systematically.

Organisations handle different types of knowledge that are often inter-related, and which must be managed in a consistent way. For instance, software engineering involves the integration of various knowledge sources that are in constant change. Therefore, tools and techniques are necessary to capture and process knowledge in order to facilitate subsequent development efforts, especially in the domain of software engineering

The changeable character of the software maintenance process requires that the information generated is controlled, stored, and shared. We propose a multi-agent system in order to manage the knowledge generated during maintenance. The roles of the these agents are summarised as follows consist of:

- Comparing new information with information that has already been stored in order to detect inconsistencies between old and new information.
- Informing other agents about changes produced.
- Predicting new client's demands. Similar software projects often require similar maintenance demands.
- Predicting possible mistakes by using historic knowledge.
- Advising solutions to problems. Storing solutions that have worked correctly in previous maintenance situations helps to avoid that due to the limited transfer of knowledge companies are forced to reinvent new practices, resulting in costly duplication of effort.
- Helping to make decisions. For instance to evaluate whether it is convenient to outsource certain maintenance activities.

– Advising certain employee to do a specific job. The system has information about each employee's skills, their performance metrics, and the projects they have worked on.

A multiagent system in charge of managing maintenance knowledge might improve the maintenance process since agents would help developers find information and solutions to problems and to make decisions, thus increasing organisation's competitiveness.

5 Discussion Topics

Based on preliminary e-mail discussions, submitted workshop papers, short presentations and discussions during the morning session of the workshop, the following relevant discussion topics were identified.

5.1 Mapping between Knowledge Representation and Object-Oriented Programs

This topic was explicitly addressed by several participants. Different kinds of mappings can be imagined:

– two separate representations with an explicit link between them
– a code-generation approach where there is only the represented knowledge from which code is generated automatically
– a symbiotic or reflective mapping between the knowledge representation language and the object-oriented language
– an approach where the knowledge representation language and the object-oriented language are actually one and the same language. (Note that this is more than mere symbiosis: the languages have become –or were designed to be – one and the same.)

5.2 How a Knowledge-Based Approach can Support Maintenance of Object-Oriented Programs

Many participants addressed the topic of using a knowledge-based approach to achieve, facilitate or advance software maintenance in general and software evolution, reuse and understanding in particular.

5.3 Approaches and Techniques for Dealing with Knowledge-Based Software Engineering

A whole range of different approaches towards KBOOSE were proposed or mentioned by the various participants:

– a rule-based backward chaining approach (e.g. Prolog)
– a rule-based forward chaining approach

- a mixture of forward and backward chaining (found in many knowledge systems)
- a frame-based approach (e.g. KRS [7])
- constraint solving
- description logics (these are limited but decidable subsets of first-order logic with reasoning support)
- model checking
- an agent-based approach
- ontologies

Note that this topic is orthogonal to the previous two.

5.4 Consistency Checking

This was proposed by a number of participants as a means to support maintenance of object-oriented programs. Therefore, this topic can be seen as a subtopic of topic 5.2. Consistency can be checked between models (e.g. between different kinds of UML design diagrams) or between the represented knowledge and the implementation. If we consider source code as a model too, these two kinds of consistency checking essentially boil down to the same thing.

5.5 Which Software Engineering Tasks can Benefit from a Knowledge-Based Software Engineering Approach?

This topic is closely related to topic 5.2 as well. In fact, most participants mentioned that they used knowledge-based object-oriented software engineering to support software maintenance, reuse, evolution and understanding. Of course, knowledge-based software engineering may be useful for other tasks as well.

5.6 How can the Represented Knowledge be Used?

Again this topic is closely related to topic 5.2. Indeed, checking model consistency is an interesting way of using the represented knowledge for achieving well-designed or correct software.

5.7 Summary of the Discussion Topics

Looking back at all these discussion topics, given the fact that topics 5.5 and 5.6 are closely related to topic 5.2 and because topic 5.4 can be seen as a subtopic of 5.2, we decided to focus on the first three topics only during the group discussions. Furthermore, because of the orthogonality of the third topic with respect to the first two, we decided to discuss only topics 5.1 and 5.2. However, the two discussion groups were supposed to take a look at topic 5.3 too from the perspective of their own discussion topic. More specifically, for topic 5.1 it was important to know how the choice of a particular approach or technique might affect or influence the chosen mapping. For topic 5.2 it was important to discuss which particular techniques could or should be used or avoided in order to support software maintenance.

6 Group Discussions

6.1 Mapping between Knowledge Representation and Object-Oriented Programs

This group consisted of Maja, Ellen, María Agustina, Theo, Andreas and Marie.

Interests of the participants in this topic comes both from a technology and a problem-oriented point of view. Based on all the individual interests in this particular discussion topic, we came up with the following issues:

domain and configuration knowledge In software applications many kinds of implicit knowledge can be detected. Maja and María Agustina both want to represent domain knowledge and in particular business rules explicitly and decoupled from the core application. Ellen wants to represent knowledge in the context of families of applications (e.g. frameworks or product lines) in vertical markets, among others knowledge about the domain, about variabilities in the domain (also referred to as *delta's*), and about the link between the variabilities and the implementation for traceability. An appropriate representation for these kinds of knowledge can be found in the older hybrid knowledge representation systems which combine frame-based and rule-based representations. The core application is implemented using an object-oriented programming language. Hence the need arises for a symbiosis between frame-based and rule-based representation on the one hand, and object-oriented programming on the other.

task knowledge Andreas is concerned with the task knowledge, in other words the workflow or process, in software applications. Marie is working in the domain of medical diagnosis and treatment. They both observed that the task of a software application crosscuts the entire implementation and thus becomes hard to maintain and reuse. Andreas remarks that it boils down to explicitly representing component interactions, and that possibly temporal logic would be a suitable medium. Issues that arise here are the ever ongoing discussion of black-box versus white-box components, the level of granularity of the interactions, and how to reverse engineer existing components to obtain this representation of the interactions.

Symbiosis between Declarative and Object-Oriented Languages The needs and interests of the participants pointed to a symbiosis between a declarative language and an object-oriented language. A language symbiosis means that the mapping of the declarative language onto the object-oriented language is implemented and integrated in the latter. We must note, however, that *declarative* is a paradigm rather than a language, and in order to discuss a symbiosis between two languages, a concrete declarative language must be considered. Possible candidates are:

- a truly declarative rule-based language, which means that the rule chaining strategy is transparent, i.e. forward or backward chaining, or both

– a constraint language with a constraint solver or truth maintenance system
– description logic with reasoning for ensuring consistency or classification

The main problem with a symbiosis between a declarative and an object-oriented language is consolidating the behavioural aspect: message passing and for instance rule chaining is hard to map. This is easily illustrated when one considers a message send which is in fact mapped to a rule, where there is a mapping between the parameters (including the receiver) of the message and the variables of the rule. When the rule returns more than one result, the question is how the object-oriented layer will deal with this since it only expects one result from a message.

Theo mentioned by means of a concrete example that the "holy grail" of a perfect integration of a knowledge-based language with an object-oriented language may be very hard to find. If one considers SQL, everyone will probably agree that this is a declarative language which is very well-suited for reasoning about knowledge stored in databases, but even this one is still not well-integrated with imperative or object-oriented languages.

A possible strategy for facilitating the symbiosis might be to do a co-design from scratch of a new object-oriented and declarative language that perfectly integrates the two paradigms. But then the problem probably arises that the language will not be generally accepted because it would be yet another "academic" language.

Existing Systems with Language Symbiosis were considered in this discussion. We can distinguish between systems that provide a symbiosis between two languages of the same paradigm, such as Agora [10], and systems or languages that unite multiple paradigms. Another distinction is the way the multiple paradigms are combined. On the one side of the spectrum there are the multiparadigm languages such as C++ [2] and LISP. Then there are the systems that implement one paradigm in a language based on another paradigm, which in most cases makes the underlying language accessible in the language it implements, and may even allow metaprogramming about the underlying language in the language it implements. Examples are SOUL (Smalltalk Open Unification Language), a logic programming environment implemented in Smalltalk [9]; SISC (Second Interpreter of Scheme Code) [11] and JScheme [1] both Scheme interpreters written in Java.

6.2 How a Knowledge-Based Approach can Support Maintenance of Object-Oriented Programs

This group – consisting of Kim, Miro, Atsushi, Ragnhild, Dirk, Aurora and Elke – addressed discussion topic 5.2 (including topic 5.4 as a subtopic), which was rephrased as follows:

> *How can a knowledge-based approach support software maintenance (including evolution, reuse and understanding) and what particular tech-*

niques and approaches might be helpful in doing so (for example, consistency checking among models)?

To structure the discussion, every member of the group was asked to explain and discuss (in the context of knowledge-based software engineering)

1. what they understood by *consistency checking*, and optionally how they used it and for which purpose
2. what *representation* they used to represent their knowledge and how the represented knowledge was actually *connected* to the underlying object-oriented programs

After this, the different answers were merged which lead to the following insights:

1. Consistency checking boils down to *checking whether there are no contradictions between models*, where the implementation (i.e. source code) can also be seen as a particular kind of model. Consistency checking is often syntactic but can be semantic as well (an example of the latter was Ragnhild's approach where the different models to be checked are mapped to the same description logic and then the reasoning tools in that logic are used to check for contradictions between the produced descriptions). All participants agreed that maintenance and reuse are the main goals of consistency checking. But this does not imply that consistency checking is necessary for maintenance and reuse.
2. Regarding the representation used to represent the knowledge we discovered two different kinds of approaches. The difference seemed to be caused by the different goals to support maintenance that were followed:
   - One goal was mainly to *collect information throughout the software development life cycle* mainly with the intention of consolidating the knowledge that is in different people's heads, thus contributing to a better understanding of the domain. Both Dirk's and Aurora's approach were examples of this. In Dirk's case he represented the knowledge using an ontology expressed in a frame-based language. In Aurora's case, she used an agent-based approach.
   - A second goal was to *check conformance between specific models* with the goal of improving the quality and correctness of the software. Kim's, Ragnhild's and Atsushi's approaches were examples of this. In all these cases the knowledge was represented using a logic language (either a Prolog-like language or a more restricted description logic) and a kind of logic reasoning was used to check for consistency. In the ideal case where the models to be checked were well-defined, perhaps with some extra user-defined annotations attached to them, the corresponding logic expressions can be generated and checked automatically. In the less ideal case, model checking first requires us to describe the logic expressions either manually, semi-automatically or based on heuristics.

7 Conclusions

Although the main goals were rather well addressed and we are generally pleased with the results, one has to bear in mind that it was but the first workshop on Knowledge-Based Object-Oriented Software Engineering. Therefore, we passed what one could call the exploratory phase: all topics and lines of thought that came up were considered and elaborated upon, but we did not investigate every nook and cranny of this field. For a possible next workshop related to this one, it would be interesting to make a general classification of which particular software engineering tasks may benefit from a particular knowledge-based technique or approach. The results of this query could then very well serve as a starting point for more specific in a series of workshops on Knowledge-Based Object-Oriented Software Engineering.

References

[1] Ken Anderson, Tim Hickey, and Peter Norvig. The jscheme web programming project. `http://jscheme.sourceforge.net/jscheme/mainwebpage.html`.

[2] James O. Coplien. *Multiparadigm Design For C++*. Addison-Wesley, 1998.

[3] Dirk Deridder. A concept-oriented approach to support software maintenance and reuse activities. In *5th Joint Conference on Knowledge-Based Software Engineering (JCKBSE)*. IOS Press - Series "Frontiers in Artificial Intelligence and Applications", 2002.

[4] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[5] W.L. Hürsch and C.V. Lopes. Separation of concerns. Technical report, North Eastern University, 1995.

[6] A. Kleppe and J. Warmer. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1999.

[7] Kris Van Marcke. *The Knowledge Representation System KRS and its Implementation*. PhD thesis, Vrije Universiteit Brussel, 1988.

[8] Kim Mens, Tom Mens, and Michel Wermelinger. Supporting object-oriented software development with intentional source-code. In *Proceedings of the 15th Conference on Software Engineering and Knowledge Engineering (SEKE)*. Knowledge Systems Institute, 2002.

[9] Kim Mens, Isabel Michiels, and Roel Wuyts. Supporting software development through declaratively codified programming patterns. In *Proceedings of the 14th Conference on Software Engineering and Knowledge Engineering (SEKE)*. Knowledge Systems Institute, 2001.

[10] Wolfgang De Meuter. The story of the simplest mop in the world, or, the scheme of object-orientation. *Prototype-Based Programming (eds: James Noble, Antero Taivalsaari, and Ivan Moore)*, 1998.

[11] Scott G. Miller. Second interpreter of scheme code. `http://sisc.sourceforge.net/`.

[12] The Object Management Group. *The OMG Unified Modeling Language Specification*. `http://www.omg.org`.

[13] Atsushi Ohnishi. A supporting system for verification among models of the uml. *Systems and Computers in Japan*, 33(4):1–13, 2002.

[14] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[15] A. Th. Schreiber, J. M. Akkermans, A. A. Anjewierden, R. de Hoog, N. R. Shadbolt, W. Van de Velde, and B. J. Wielinga. *Knowledge Engineering and Management: The CommonKADS Methodology*. MIT Press, 2000.

Participant Names, E-Mail Addresses, and Affiliations

name	`e-mail` and affiliation
Marie Beurton-Aimar	`Marie.Aimar@u-bordeaux2.fr` Université Bordeaux 2 - France
Miro Casanova	`mcasanov@vub.ac.be` Vrije Universiteit Brussel - Belgium
María Agustina Cibrán	`mcibran@vub.ac.be` Vrije Universiteit Brussel - Belgium
Krzysztof Czarnecki	`czarnecki@acm.org` Daimler Chrysler - Germany
Dirk Deridder	`Dirk.Deridder@vub.ac.be` Vrije Universiteit Brussel - Belgium
Maja D'Hondt	`mjdhondt@vub.ac.be` Vrije Universiteit Brussel - Belgium
Theo D'Hondt	`tjdhondt@vub.ac.be` Vrije Universiteit Brussel - Belgium
Kim Mens	`Kim.Mens@info.ucl.ac.be` Université catholique de Louvain - Belgium
Atsushi Ohnishi	`ohnishi@acm.org` Ritsumeikan University - Japan
Elke Pulvermueller	`pulvermu@ipd.info.uni-karlsruhe.de` Universitaet Karlsruhe - Germany
Andreas Speck	`a.speck@intershop.com` Intershop Research - Germany
Ragnhild Van Der Straeten	`rvdstrae@vub.ac.be` Vrije Universiteit Brussel - Belgium
Ellen Van Paesschen	`evpaessc@vub.ac.be` Vrije Universiteit Brussel - Belgium
Aurora Vizcaíno	`avizcaino@inf-cr.uclm.es` Universidad de Castilla La Mancha - Spain