A Framework for Ontology Reuse and Persistence Integrating UML and Sesame

C. Pedrinaci, A. Bernaras, T. Smithers, J. Aguado, M. Cendoya

San Sebastian Tecnology Park, Paseo Mikeletegi 53,
20009 San Sebastian, Spain
{carlos, amaia, tsmithers, jessica}@miramon.net
mcendoya@miramon.es

Abstract. Nowadays there is a great effort underway to improve the World Wide Web. A better content organisation, allowing automatic processing, leading to the Semantic Web is one of the main goals. In the light of bringing this technology closer to the Software Engineering community we propose an architecture allowing an easier development for ontology-based applications. Thus, we first present a methodology for ontology creation and automatic code generation using the widely adopted CASE UML tools. And based on a study of the art of the different RDF storage and querying systems, we couple this methodology with the Sesame system for providing a framework able to deal with large knowledge bases.

1 Introduction

The huge amount of information available in the World Wide Web has led researchers to work towards improving its organisation, by providing machine-understandable data. "*The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.*"[1]. It is obvious that the Semantic Web will offer new possibilities for the web but as Mark Frauenfelder suggests "*There is a big question as to whether people will think the benefits are worth the extra effort of adding metadata to their content in the first place. One reason the Web became so wildly successful, after all, was its sublime ease of creation.*"[2].

This paper presents some of the results obtained in the ongoing EU project OBELIX (IST-2001-33144) during the creation of an ontology-based online events design application [3],[4]. We propose a framework for the development of Semantic Web applications development so as to bring this technology closer to the Software Engineering community. Bearing that purpose in mind, the proposed framework focuses on the ease of creation and use. The same way web designers don't have to be aware of HTTP protocol's details (and very often even of HTML details), it would be interesting to obtain the same level of independency from the implementation details surrounding the Semantic Web which are much more complex. Obtaining such facilities for creating Semantic Web applications is difficult due to its inherent complexity, but we should however try to fill the gap between AI community and the Software Engineering community, by providing an easy and suitable framework.

Moreover, software agents will need to interact with other systems, usually based on different ontologies, supported by different architectures and adequately supporting the interaction with humans. Semantic Web applications are complex systems, thus, maintaining and/or improving them is a hard task. Software Engineering has proven that in such cases, and in general for every system, a clear, well defined and powerful methodology is a must. Such methodologies facilitate the creation and minimize the problems raised when improving and modifying a system.

In this paper, we first present the use of the Unified Modelling Language (UML) [5] for knowledge representation, along with a procedure for generating from a UML class diagram a specialised RDF schema [6],[7] and a set of Java classes corresponding to the classes in the model. Afterwards we compare the different RDF storage and querying systems, and justify the selection of Sesame for ensuring persistance for large RDF knowledge bases [8]. Next, we present and explain Sesame. In section five, we propose an architecture for developing ontology-based applications, using UML for knowledge representation and relying on Sesame for data persistance. Finally, we conclude and present some directions for future research.

2 UML for Knowledge Representation and Exchange

The Unified Modeling Language (UML) is a standard language from the Object Management Group (OMG) [9] with an associated graphical notation for object-oriented analysis and design. It is widely adopted in industry, and several CASE tools are already available to facilitate software engineers' work. The benefits of using UML for ontology development have been extensively argued in [10], [11], [12] and [13]. Some of these benefits are: (i) UML is a standard language; (ii) UML is a graphical notation based on many years of experience in software analysis and design, which is currently suported by widely-adopted CASE tools that are more accessible to software practitioners than current ontology tools; (iii) agent-based systems will need to interact with legacy enterprise systems, which often have UML models; (iv) knowledge expressed using UML is directly accessible for human comprehension and for machine processing; (v) thanks to the modular nature of object-oriented modelling, the knowledge in a UML model can be changed without affecting other features.

In [11] and [13] Stephen Cranfield proposes an implementation for object-oriented knowledge representation, using UML for defining ontologies and domain knowledge in the Semantic Web. Fig. 1 shows a pictorical description of this proposal. The proposed methodology is as follows. First, a domain expert designs the ontology graphically with one of the available CASE tools supporting UML (e.g. Rational Rose, Poseidon, ArgoUML, etc). The ontology is then saved in the standard format XML Model Interchange (XMI) [14]. Using a pair of XSLT stylesheets the XMI representation of the ontology is transformed into a set of Java classes and interfaces corresponding to the concepts present in the ontology, and into an RDF schema. The java classes allow an application to represent knowledge about the domain as in-memory data structures. The RDF schema, defines the concepts that an application can reference when serializing the knowledge in RDF/XML. For performing the marshalling and unmarshalling of objects to and from RDF/XML documents, a mar-

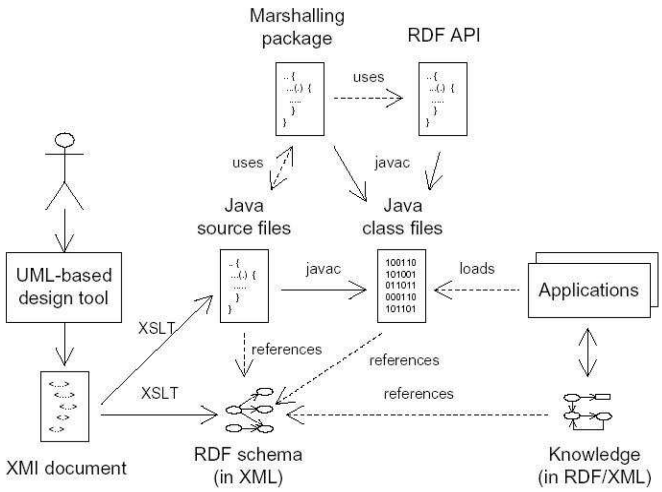

Fig. 1. Overview of the imlementation for object-oriented knowledge representation. (Taken from [13])

shalling package is also provided. This feature is provided via two classes: MarshalHelper and UnmarshalHelper. These delegate to the generated Java classes decisions about the names and types for each field, and are then called back to perform the un/marshalling from/to RDF, using the Stanford RDF API [15].

It is important to note that the generated RDF schema does not contain all the information from the designed UML model. Its purpose is to define resources corresponding to all the classes, interfaces, attributes and associations in the ontology in order to allow serialisation of in-memory objects in the standard language RDF. Thus, for accessing all the ontology information one of the available Java APIs for XMI can be used: [16] and [17].

The system does also allow modelling incomplete knowledge. Therefore, the generated Java classes include extra boolean fields for each attribute that record whether the value is known or not. Also, when marshalling incomplete information, a non-standard RDF property, *notClosedFor,* is used and associates a property with a resource, meaning that the information is incomplete.

Obtaining an instance from the RDF/XML representation involves parsing the whole file, which is not a problem for small knowledge bases. However, when dealing with large knowledge bases, there are more efficient approaches: RDF storage and querying systems.

3 Comparison of the RDF Storage and Querying Systems

To adapt Cranefield's approach to large knowledge bases, we have studied the different RDF storage and querying facilities available. The state of the art of the different systems is based on [18], with updated information.

Table 1 presents an analysis of the different storage systems currently available. The main criteria that were kept in mind for determining the RDF storage and querying system that suits better to our needs are:

- Storage: The method/architecture used for ensuring the data persistance.
- Platform: List of all the different platforms supported. It includes the Operating Systems but also the need for any other components like a Perl interpreter or a Java Virtual Machine.
- API: The possible ways for interacting with the system. It includes protocols and APIs provided for different programming languages.
- Querying: The languages the system allows to be used for querying a data repository.
- Inferencing: The capability of the system to infer new knowledge, that is to generate new statements based on the existing knowledge. For the majority of the systems only class subsumption is provided. However, some systems allow more powerful inferencing by providing mecanisms for defining user rules.
- Extras: Whether the system has other functional elements associated or prepared for interacting with it.

From the analysis and comparison performed, and shown in , Sesame was chosen for the following reasons:

Sesame allows inferencing over RDF(s) thanks to its query language RQL [19], [20]. Moreover, the system can be deployed in any platform with a Java Virtual Machine. It provides several ways for interacting with it such as RMI, SOAP or HTTP. It has been installed on top of many DBMS like Oracle, MySQL or PostgreSQL and has a generic implementation for SQL92 compliant DBMS. In addition to all these characteristics, support for DAML+OIL [21] has been added, improving its capabilities but also showing Sesame's modularity and the possibility to adapt the system to new languages. Finally, the new versionning and access control features implemented, turn Sesame into a suitable system for developing and maintaining knowledge bases providing the same control level as CVS does for programmers.

It is worth noting that, although KAON [22] and Cerebra [23] are good candidates for their interesting features, Sesame is superior to KAON for its support for DAML+OIL. Concerning Cerebra the fact it is not Open Source was determinant.

4 Sesame

Sesame is a system for efficient storage and expressive querying of large quantities of metadata in RDF and RDF Schema. It was initially developed by Aidministrator Nederland b.v. as part of the European IST project On-To-Knowledge [24] and is currently been extended and improved by Aidministrator Nederland b.v., the "Sesame community" and NLNet [25].

Table 1. RDF storage and querying systems comparison

	Querying	*Platform*	*API*	*Storage*	*Inference*	*Extras*
ICS-RDFSuite	RQL	Solaris - Linux	Java - C++ - SQL	O-RDBMS	Yes	
Sesame	RQL*, RDQL	Any (JVM)	Java - HTTP RMI - SOAP	O-RDBMS	Yes	OMM, BOR
Inkling	SquishQL	Any (JVM)	Java	Memory - JDBC	No	
RDFDB	SquishQL*	Solaris - Linux - FreeBSD	C - Perl	SleepyCat	Yes	
RDFSTORE	SquishQL	Any (Perl)	Perl	Memory - BerkeleyDB	Yes	
EOR	Triple-matching	Any (JVM)	Java - HTTP SQL	SQL DB	No	
Redland	Triple-matching	Solaris - Linux - MacOS X FreeBSD - OSF/1	C - Java - Perl Python - Tcl	Memory - SleepyCat BerkeleyDB	No	
Jena	RDQL	Any (JVM)	Java	Memory - BerkeleyDB PostgreSQL	No	
RDF Gateway	RDFQL	Windows NT/2000	ADO - JDBC	RDBMS	Yes (+ user-defined)	
TRIPLE	TRIPLE	Any (JVM)	Java	Memory	Yes (+ user-defined)	RACER
KAON	F-Logic	Any (JVM)	Java	Memory - RDBMS	Yes (+ user-defined)	KAON TOOL SUITE
CEREBRA	DL-based	Any (JVM)	Java - SOAP	Distributed data (CORBA)	Yes (+ user-defined)	Cerebra Suite

"Sesame's design and implementation are independent from any specific storage device. Thus, Sesame can be deployed on top of a variety of storage devices, such as relational databases, triple stores, or object-oriented databases, without having to change the query engine or other functional modules" [8]. This independence is granted by the *Storage And Inference Layer* (SAIL) (see Fig. 2). SAIL is an Application Programming Interface (API) that offers specific methods for accessing RDF information. It defines a basic interface for storing, retrieving and deleting RDF and RDFS from repositories while it abstracts from the particular storage mechanism. It was designed to support low end hardware like PDAs and to be extendable to other RDF-based languages. Several implementations of SAIL are distributed with Sesame like SQL92SAIL, which is a generic implementation for SQL92 compliant DBMS, Sync-SAIL for supporting concurrent reads as well as implementations for specific DBMS like MySQL, OracleDB and PostgreSQL.

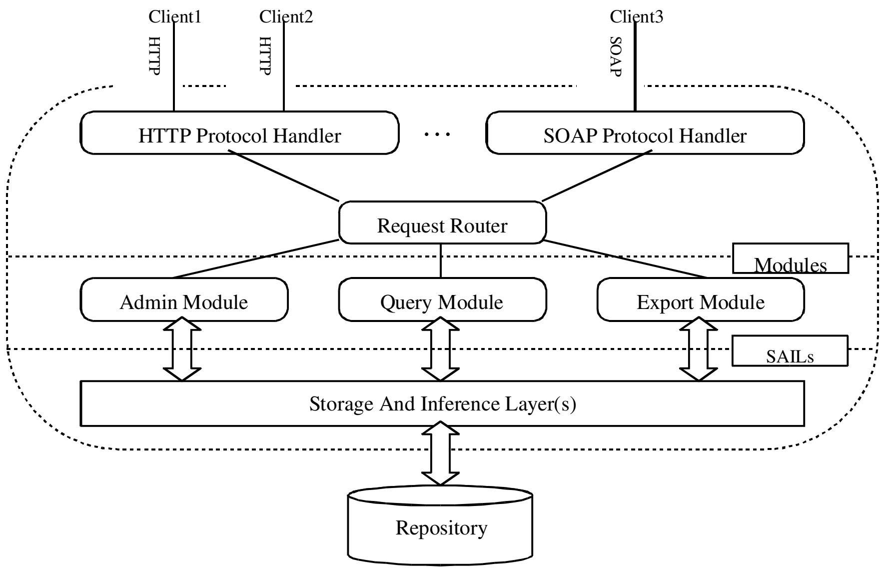

Fig. 2. Sesame's architecture. Taken from [8]

Sesame implements the Resource Query Language (RQL) a declarative language for querying both RDF descriptions and RDF schemas, as well as RDQL [26] which is derived from SquishQL [27]. These functions are provided by the *Query Module* which performs the queries on a repository. Any query is first parsed to build a tree model representation, which is afterwards optimised. The majority of the query is evaluated in this module, the access to the repository is handled by SAIL. It is important to note that Sesame implements a slightly modified version of the RQL language proposed in [20]. Sesame's version of RQL includes support for domain and range restrictions as well as multiple domain and range restrictions, but it does not feature support for datatyping.

For the metadata administration, another module is provided, the *Admin Module.* Its purpose is to manage the insertion and deletion of RDF and RDF Schema information into/from a repository.

The extraction of any information from a Sesame repository is handled by the *Export Module*. This module allows to selectively export the schema, the data or both from a repository, facilitating the integration and interaction with other RDF tools.

Concerning the interaction with external applications Sesame currently offers three methods: HTTP, SOAP and RMI. Each protocol has its associated handler, which translates and redirects any query received into an intermediate module: the *Request Router*. This intermediate module abstracts Sesame's core from any protocol specificity leaving the possibility to add a new handler without having to modify the rest of the system.

For making the results of the On-To-Knowledge project easier for integration in real-world applications an "administrative" software infraestructure was created: The Ontology Middleware (OMM). "*The central issue is to make the methodology and modules available to the society in a shape that allows easier development, management, maintenance, and use of middle-size and big knowledge bases"*[28]. In particular the OMM supports versionning, access control and meta-information for knowledge bases forming the Knowledge Control System (KCS). In addition to the administrative modules, BOR extends the reasoning capabilities of Sesame by providing support for DAML+OIL. This new reasoning module implements the SAIL API, thus it can perfectly interact with the rest of the modules of Sesame.

5 Architecture Proposal

We have seen previously that in Stephen Cranefield's approach a marshalling package is used for mashalling and unmarshalling object-oriented information between in-memory data structures and RDF serialisations of that information. This solution is not efficient enough for managing large RDF files. Thus, the available RDF storage and querying tools have been studied, and Sesame was choosen based on its characteristics.

In order to support large knowledge bases (more than five thousand triples), Fig. 3 shows an adaptation of Stephen Cranefield's approach by replacing the marshalling elements by calls to the Sesame API. Any serialisation or deserialisation of knowledge is performed over an RDF repository in Sesame. The generation of ontology-based applications remains, from the developer point of view, unchanged and transparent. The process still involves editing the ontology in a CASE environment supporting UML and XMI. Afterwards Java classes and the RDF Schema file are generated and their usage, during the creation of an ontology-based application, remains unmodified. However, the architecture gains greatly in versatility and power due to the new mechanisms that grant the persistence and access of the knowledge base provided by Sesame. The RDF/RDF Schema is stored in a Sesame repository. Thus, applications interact with Sesame for retrieving and/or storing knowledge and at the same time they have all the Sesame's features available like, for example, the querying language RQL.

There is however an important difference concerning the generation of the Java classes. The proposed architecture maintains the XSLT for generating the RDF schema file, whereas the generation of the Java classes is not performed using XSLT. We are developing a Java Code Generator that benefits from Sesame's features by accessing the ontology stored in a Sesame repository where the associated RDF schema has been stored. Thanks to the SAIL API Sesame offers, our program can browse the whole ontology in a more confortable way. Thus, the difficulties associated to the use of a stylesheets processor are avoided. Moreover, the code generation gains in modularity, and ease of maintenance, so that future improvements can be easily added.

We are also investigating another important aspect, which is the possibility of adapting the whole system to a more powerful language like DAML+OIL. Several projects are already using UML and DAML+OIL together. The UML Based Ontology Tool-set (UBOT) project [29] is working on an UML to DAML mapping [30]. In this project UML is also used as a front-end for visualizing and editing DAML ontologies. Also, the Components for Ontology Driven Information Push (CODIP) project [31] is using UML to build and map DAML ontologies. This project is creating the DAML-UML Enhanced Tool (DUET) which provides a UML visualization and authoring environment for DAML. Core DAML concepts are being mapped into UML through a UML profile for DAML. DUET is currently available as a plug-in for Rational Rose [32] and ArgoUML [33]. The results of both projects could be applied to the proposed architecture for obtaining a “DAML+OIL version”.

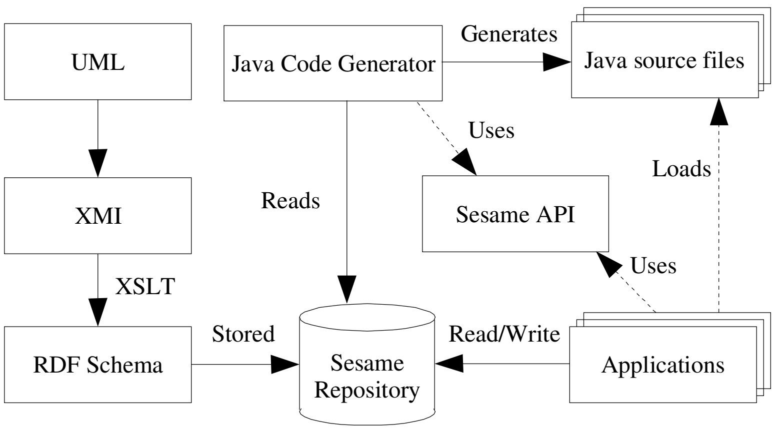

Fig. 3. Architecture proposal

Finally, in addition of the persistance related benefits, Sesame comes with a Web interface, that can be installed on a web server like Tomcat. This is a step forward for publishing the ontologies along with the instances in the World Wide Web, so that external applications like agents, can also retrieve the information and process it.

6 Conclusions and Future Research

In the previous sections we have described an architecture for creating ontology-based applications in a more suitable way for Software Engineers than the currently available tools like OilED, Ontoedit or Protégé. This architecture integrates the UML to RDF mapping based on the approach presented in [11] and [13], with Sesame as the RDF storage and querying system. This integration is also improved by the addition of our Java Code Generator which makes use of the best of the two integrated approaches. The result is a framework for developing ontology-based applications in an easy and scalable way, with an automatic code generation to facilitate the use of object diagrams as internal knowledge representation structures. However, the majority of the ontology-based applications that have been developed, have shown that an ontology expressed in RDF or DAML+OIL is not enough for obtaining all the needed functionality. They still need the capability to define rules and constraints, so as to provide more powerful inferencing over the knowledge base. Unfortunately there is no standard language for defining rules. This has been solved by different developers with ad hoc methods: choosing the most appropriate and convenient inferencing engine or directly with hard-wired code. In our case, there is no mechanism provided for defining inferencing rules, thus it would be desirable to cover also that aspect. UML's definition includes the Object Constraint Language (OCL), however it lacks a formal definition. Currently the precise UML group [34] is addressing this issue.

With a formal specification, the code generation could also integrate automatic rules generation based on the OCL rules definition. This kind of code generation has already been undertaken by Frank Finger in [35]. Further research is needed in that respect.

Finally, we are also investigating dynamic code generation over evolving ontologies so as to provide a better adaptability to the dynamism of the Web.

References

1.T. Berners-Lee, J. Hendler and O. Lassila: The Semantic Web. Scientific American (2001)

2.M. Frauenfelder: A Smarter Web. Technology Review (2001)

3.M. Cendoya, A. Bernaras, T. Smithers, J. Aguado, C. Pedrinaci, I. Laresgoiti, E. García, A. Gómez, N. Peña, A. Z. Morch, H. Sæle, B. I. Langdal, J. Gordijn, H. Akkermans, B. Omelayenko, E. Schulten, J. Gordijn, B. Hazelaar, P. Sweet, H.-P.Schnurr, H. Oppermann, and H. Trost: D3 Business needs, Applications and Tools Requirements (2002)

4.A. Maier, J. Aguado, A. Bernaras, I. Laresgoiti, C. Pedinaci, N. Peña,T. Smithers: Integration with Ontologies. 2nd Conference on knowledge management (WM2003) (2003)

5.OMG: Unified Modelling Language Specification version 1.5 (2003)

6.D. Brickley, R.V. Guha: Resource Description Framework(RDF) Schema Specification 1.0. http://www.w3.org/TR/2000/CR-rdf-schema-20000327 (2000)

7.O. Lassila, R. R. Swick: Resource Description Framework(RDF) Model and Syntax Specification. http://www.w3.org/TR/REC-rdf-syntax/ (1999)

8.J. Broekstra, A. Kampman, and F. van Harmelen: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. International Semantic Web Conference (ISWC) (2002)

9. Object Management Group web page. http://www.omg.org. (Last visited: June 2003)
10. S. Cranefield and M. Purvis: UML as an Ontology Modelling Language. In Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99) (1999)
11. S. Cranefield: Networked Knowledge Representation and Exchange using UML and RDF. Journal of Digital Information (2001)
12. P. Kogut, S. Cranefield, L. Hart, M. Dutra, K. Baclawski, M. Kokar, J. Smith. UML for Ontology Development. Knowledge Engineering Review Journal Special Issue on Ontologies in Agent Systems (2002)
13. S. Cranefield: UML and the Semantic Web. Proceedings of the International Semantic Web Working Symposium (2001)
14. Object Management Group: OMG XML Metadata Interchange (XMI) Specification (2002)
15. Sergey Melnik: RDF API. http://www-db.stanford.edu/~melnik/rdf/api.html. (Last visited: June 2003)
16. Unisys Corporation: Java Metadata Interface (JMI) specification (2002)
17. Novosoft: Novosoft metadata framework and UML library (2002)
18. A. Magkanaraki, G. Karvounarakis, T. T. Anh, V. Christophides, D. Plexousakis: Ontology storage and querying. Technical Report 308, ICS-FORTH (2002)
19. G. Karvounarakis, V. Christophides: The RQL v1.5 User Manual. http://139.91.183.30:9090/RDF/RQL/Manual.html . (Last visited: June 2003)
20. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl: RQL: A Declarative Query Language for RDF. 11th International World Wide Web Conference (WWW2002) (2002)
21. Joint United States / European Union ad hoc Agent Markup Language Committee: DAML+OIL (March 2001) release (2001)
22. KAON web page. http://kaon.semanticweb.org/ . (Last visited: June 2003)
23. Cerebra home page. http://www.networkinference.com/. (Last visited: June 2003)
24. On-To-Knowledge (IST-1999-10132) web page. http://www.ontoknowledge.org/ .(Last visited: June 2003)
25. Sesame Project web page. http://sourceforge.net/projects/sesame/. (Last visited: June 2003)
26. RDF Data Query Language (RDQL). http://www.hpl.hp.com/semweb/rdql.htm. (Last visited: June 2003)
27. L. Miller: RDF Squish query language and Java implementation. http://www.ilrt.bris.ac.uk/discovery/2001/02/squish/. (Last visited: June 2003)
28. A. Kiryakov, K. Simov, D. Ognyanov: Ontology Middleware: Analysis and Design. IST Project IST-1999-10132 On-To-Knowledge Deliverable 38 (2002)
29. UBOT web page. http://ubot.lockheedmartin.com/. (Last visited: June 2003)
30. K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes, J. Letkowski, M. Aronson: Extending UML to Support Ontology Engineering for the Semantic Web. 4th International Conference on UML (2001)
31. DARPA, AT&T: Components for Ontology Driven Information Push (CODIP) Home Page. http://codip.grci.com/. (Last visited: June 2003)
32. Rational UML software home page. http://www.rational.com/uml/index.jsp. (Last visited: June 2003)
33. ArgoUML web page. http://argouml.tigris.org/. (Last visited: June 2003)
34. Precise UML group home page. http://www.puml.org/. (Last visited: June 2003)
35. Frank Finger: Design and Implementation of a Modular OCL Compiler. Diploma Thesis (2000)