

Open Reuse of Component Designs in OPM/Web

Iris Reinhartz-Berger, Dov Dori, and Shmuel Katz
Technion - Israel Institute of Technology
Technion City, Haifa, Israel 32000
{[ieiris@tx](mailto:ieiris@tx.technion.ac.il), [dori@ie](mailto:dori@ie.technion.ac.il), [katz@cs](mailto:katz@cs.technion.ac.il)}.technion.ac.il

Abstract

As system complexity has increased, so has interest in reusing software components in early development phases. While most current modeling methods support design of generic parameterized frameworks or patterns and weaving them into specific models, they do not support open reuse, i.e., the ability to develop partially specified components and refine them in the target application. We introduce an open reuse formalism that is based on OPM/Web, an extension of Object-Process Methodology for distributed systems and Web applications. Our open reuse is accomplished by a three-step process, consisting of designing reusable models, creating basic woven models, and enhancing their specification. We model a reusable component through partially specified environmental elements that are bound to concrete counterparts when the component is integrated into the system under development. Rules for modeling and combining components are defined and applied to a Web example.

1. Introduction

There is widespread interest in the use of existing software artifacts or knowledge to create new software [11]. Such software reuse aims at improving software quality and productivity by integrating existing components, such as commercial off-the-shelf (COTS) products or tested modules from other projects. Early software reuse concerned the combination of reusable source code components to produce application software [14]. The object-oriented paradigm highlighted reusability as part of the development process by using classes, packages (modules), and the inheritance mechanism as primary linguistic vehicles for reuse [3]. The current definition of software reuse encompasses all the resources used and produced during the development process,

including reuse of requirements, architecture, design, implementation, and documentation.

We distinguish two reuse categories: closed and open reuse. *Closed reuse* incorporates existing complete, stand-alone modules or components, such as packages or classes, into new applications via the components' interfaces [2]. In *open reuse*, we integrate partially specified components into one combined model. Like closed reuse, open reuse encompasses the design of reusable models and their integration with the system under construction. In addition, open reuse enables enhancing and optimizing the integrated components during the analysis and design phases.

Most software engineering methods support closed reuse by parameterization and binding capabilities, but do not deal with open reuse. We suggest a formalism for both closed and open reuse using OPM/Web [15], an extension to the Object-Process Methodology (OPM) for distributed systems and Web applications. OPM [6, 7] extends the object-oriented paradigm with a new entity, called *process*, which is a pattern of transformation (consumption, generation, or change) that objects undergo. This provides OPM with a straightforward way for specifying stand-alone processes, which are not owned by (or encapsulated in) a specific object and therefore cannot abide by the object-oriented encapsulation principle. Section 2 reviews existing reuse specification techniques and argues that they primarily handle closed reuse. Section 3 specifies rules for weaving components in OPM/Web and discusses the semantics of the resulting models, while Section 4 demonstrates the process of weaving models on a Web-based example.

2. Reuse of component designs in modeling

Current object-oriented approaches, most notably UML [13, 16], emphasize the importance of reuse during the development process and enable it through classes, packages, and the inheritance mechanism. However, once the classes and packages have been modeled, they are treated as closed, black boxes with interfaces, through

which other parts of the model or other models can communicate. This approach hinders reusing generic models in different contexts. To respond to this challenge, AOP [4] introduces the concept of aspect, which modularizes the features for a particular concern and describes how these features should be woven, i.e., incorporated and integrated, into the system model. Superimposition language constructs [12] similarly extend the functionality of process-oriented systems, again cutting across the software architecture.

Recent attempts have been made to extend the aspect notion from programming to software design and engineering [1]. Most aspect-oriented modeling is based on UML, adding stereotypes to model the new aspect-oriented concepts. Catalysis [8] is an OMG-compliant methodology for component and framework-based development. Troll [9] and Composition Patterns [5] suggest adding parameterization and binding capabilities to UML packages. While these methods handle closed reuse, they lack the ability to support the development of several components into complete systems. Such support is often essential for optimizing and enhancing the design of an entire model – a mission that goes beyond binding existing components together.

Object-Process Methodology (OPM) [6, 7] is a suitable basis for open reuse, since it integrates concepts from the object- and process-oriented approaches within a single frame of reference. The elements of the OPM ontology are things and links. A *thing* is a generalization of an *object* and a *process* – the two basic building blocks of any system expressed in OPM. Analogously, links can be structural or procedural. *Structural links* express static relations between pairs of objects, such as aggregation and generalization. *Procedural links* connect objects and processes to describe the behavior of a system – how processes transform objects. OPM manages complex system models through two refinement/abstraction mechanisms: *unfolding/folding*, which is used for detailing/abstracting the structural parts of a thing, and *in-zooming/out-zooming*, which exposes/hides the inner details of a thing within its frame. This way OPM enables specifying a system to any desired level of detail without losing legibility and comprehension of the resulting model. The code generated from an OPM model describes the system dynamics, not just its skeleton, as is often the case with other modeling approaches. The main limitation of OPM with respect to reusability is that it is not geared towards the integration of a number of semi-specified components into one model of a complete application.

3. Weaving OPM/Web models

OPM/Web [15] extends OPM for the domains of distributed systems and Web applications. These extensions allow characterizing links with specific

features (objects and/or processes), extending the refining mechanisms to increase modularity, separating between the definition of a process class and process instances (occurrences) of that class to model code migration, and adding global data integrity and control constraints to express dependence or temporal relations among physically separate modules. Table 1 summarizes the relevant symbols of OPM/Web.

Table 1. Relevant OPM/Web symbols

Category	Name	Symbol
Things	Systemic object	
	Environmental object	
	Systemic process	
	Environmental process	
Structural Links	Inheritance	
	Characterization	
	Aggregation	
Procedural Links	Instrument link	
	Result/consumption link	
	Systemic effect link	
	Environmental effect link	
	Condition link	
	Invocation link	

In this work, we endow OPM/Web with open reuse capability. The open reuse is achieved by a three-step process, consisting of (1) designing reusable components, (2) integrating these components to create basic woven models, and (3) enhancing the basic woven models into complete applications. While the first two steps could be partially carried out by applying UML, the third step is not supported by object-oriented methods, since their resulting woven models are structural in nature and closed through pre-defined parameters.

3.1. Designing reusable models

The specification and design of each reusable model is carried out using OPM/Web. A thing (object or process) in an OPM/Web model that needs further specification in the target system to which it may be bound is defined as environmental. In OPM, an *environmental thing* is completely external to the system, as opposed to a *systemic thing*, which is internal to the system. In OPM/Web, an environmental thing can have partly specified internal structure, which may contain both systemic and environmental elements. When components are woven into one model, each environmental thing in the combined model is either bound to a specific thing or left as an unbound requirement. A component may also include requirements related to things in the target system to which it is bound. Such requirements are expressed by environmental (dashed) links, which connect pairs of environmental things. As explained in Section 3.2, an environmental link in a reusable component requires the

existence of a corresponding link in the target system model.

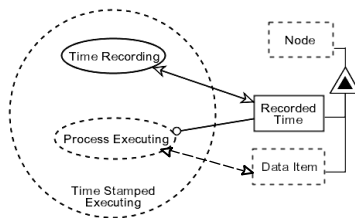


Figure 1. A reusable Time Stamp model

Figure 1 shows an example of a reusable model for adding time-stamps to a process execution. The model, called **Time Stamp**, attaches to each **Data Item** a timestamp, **Recorded Time**, and **Recorded Time** characterizes (are the attributes of) **Node**. **Data Item** is environmental since it needs to be refined and adapted to the various contexts in which this model is reused. **Recorded Time**, on the other hand, is systemic—it is internal to the model and will remain the same regardless of the context of **Data Item**. Similarly, **Process Executing** should be adapted to a specific process in the target system and is therefore denoted as an environmental process, while **Time Recording** is a systemic process. The environmental effect link between **Process Executing** and **Data Item** implies that this model can be reused only with components in which the process bound to **Process Executing** can affect (change the value of) the object bound to **Data Item**.

Any OPM/Web model is required to abide by intra-model weaving rules defined below.

The scaling rule: A systemic thing can only be refined (unfolded or in-zoomed) by other systemic things, while an environmental thing can be refined by environmental or systemic things. In Figure 1, **Time-Stamped Executing** is an environmental process, since it contains an environmental process, **Process Executing**. Similarly, the object **Node** is environmental, since one of its attributes, **Data Item**, is environmental.

The link attachment rule: Two things in a reusable component that are connected by an environmental link must both be environmental, while systemic links can connect either systemic or environmental things. In Figure 1, the environmental effect link connects an environmental object, **Data Item**, to an environmental process, **Process Executing**. If this link were replaced by a systemic one, then the things in the target model that are bound to **Data Item** and **Process Executing** would become connected by an effect link, even though this link might not be explicitly specified in their original model.

3.2. Creating basic woven models

Having created a set of OPM/Web components, the designer should decide which components are to be

woven into the target model and how to weave them. Each model, be it a reusable component or a target model, is contained in a rectangular frame, called a *module*. Usually one module is a generic reusable component and the other is the target one and is specific. The resultant woven module can be entirely concrete, or it may still contain environmental elements, which imply that the weaving process is not yet complete.

The designer can connect each environmental thing of an OPM/Web module with an environmental or systemic thing of another module. Since each module may contain things that need to be bound at different levels of refinement, the designer can successively apply the appropriate series of refining steps to get the needed design portion. The model in Figure 2, for example, weaves the model of Figure 1 into a specific **DB Maintenance** model, such that the combined specification contains two modules—the **Time Stamp Module** with **Time-Stamped Executing** in-zoomed, and the **DB Maintenance Module** with **DB Handling** in-zoomed.

Generalization-specialization relations are the primary means for binding between things in any two different modules. This relation gives rise to object-oriented inheritance by providing not only for structural object inheritance, but also for process inheritance, which is behavioral in nature. In process inheritance, the sub-process class has at least the same interface (i.e., the set of procedural links) and behavior (i.e., sub-processes) as the super-process class. The interface and behavior of the inheriting process class may be extended or restricted. This way, things in OPM/Web can inherit not just complete classes, as in UML, but also partially specified objects or processes. In both inheritance types, multiple inheritance is allowed.

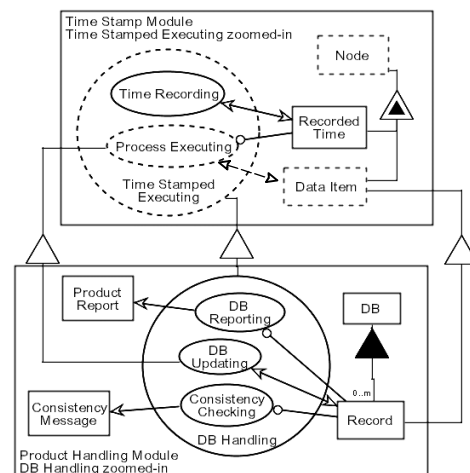


Figure 2. The Time Stamp Module woven into the DB Maintenance Module

Figure 2 shows three generalization-specialization relations: one object inheritance relation between **Data Item** and **Record**, and two process inheritance relations

one between **Process Executing** and **DB Updating** and the other between **Time-Stamped Executing** and **DB Handling**. These relations imply that **DB Updating** affects **Record** and uses **Recorded Time** as an input.

3.2.1. Inter-model weaving rules. Each woven module is required to preserve three inter-model weaving rules, in addition to the (intra-) scaling and link attachment rules.

The minimal binding rule: Each environmental thing in a module can be bound to a corresponding thing in another module, either explicitly or implicitly. A generalization-specialization relation achieves explicit binding, while implicit binding is applicable only to a compound environmental thing. A compound thing is an object or a process that is further refined and is therefore not at the lowest level of detail. For each compound environmental thing in the reusable module that is not explicitly bound to a thing in the target module, a default systemic thing is implicitly generated, including all the specific things bound to its environmental constituents. In the woven model in Figure 2, **Node** is not explicitly bound, hence a default systemic object is implicitly generated for it, including **Record**. The default object also inherits **Recorded Time** from **Node**. Likewise, if there were no process inheritance relation between **Time-Stamped Executing** and **DB Handling**, then a default process would be implicitly generated for **Time-Stamped Executing**, consisting of **DB Updating** and inheriting **Time Recording** from **Time-Stamped Executing**.

The hierarchy congruence rule: The hierarchy structure between environmental things in a reusable module must be congruent with that of the things bound to them in the target module. As an example, **Time-Stamped Executing** in Figure 2 is bound to **DB Handling**, while **Process Executing** is bound to **DB Updating**. The zooming relation between **Time-Stamped Executing** and **Process Executing** is maintained in the specific module by **DB Handling** and **DB Updating**. The hierarchy congruence rule disallows binding **Time-Stamped Executing** with **Consistency Checking** and at the same time **Process Executing** with **DB Updating**.

The link precedence rule: The binding of an environmental link is implicitly determined from the bindings of its connected things. Environmental links can be bound to systemic links which are at least as strong as the environmental according to the link precedence order.

consumption/result links are the most powerful, followed by effect links, then agent and instrument links, and finally condition and event links. The environmental effect link between **Data Item** and **Process Executing** in Figure 2 is implicitly bound to the systemic effect link between **Record** and **DB Updating**. The link precedence rule implies that a legal binding can also connect **Data**

Item and **Process Executing** with an object and a process already connected by a consumption or result link.

3.2.2. Semantics of basic woven models. The semantics of the generalization-specialization relation between modules is similar to their semantics within a single module. An object can inherit the features (attributes and operations) of a partially specified (environmental) object. Similarly, a process can inherit the behavior (sub-processes) and interface (procedural links) of a partially specified (environmental) process. In the woven model of Figure 2, **Node** has two attributes, **Recorded Time** and **Record**, where the latter inherits from **Data Item**. **DB Handling** consists of four sub-processes, but only **DB Updating** inherits from **Process Executing** an instrument link to **Recorded Time** and requires an effect link to **Record**. In other words, **DB Updating** uses **Recorded Time** as an extra input.

If different types of links exist between two environmental things and their corresponding things in a bound module, then the more powerful link (according to the link precedence order) prevails. If there were a systemic instrument link between **Data Item** and **Process Executing** in the **Time Stamp Module**, it would be subsumed by the effect link between **Record** and **DB Updating**, because of the precedence.

Generalization-specialization relations between processes also define a partial execution order between the sub-processes of an individual process. The time axis in an OPM/Web System Diagram (SD) goes from the top of the diagram to its bottom within each in-zoomed process. Hence, two independent or concurrent sub-processes are depicted at the same vertical level. The generalization-specialization relations between modules merge the partial orders from each module into one combined partial order. In Figure 2, there is a total order in both the **Time Stamp Module** (**Time Recording** and then **Process Executing**) and in the **DB Handling Module** (**DB Reporting**, then **DB Updating** and finally **Consistency Checking**). The generalization-specialization relation between **Process Executing** and **DB Updating** defines a partial execution order in the woven model: first **Time Recording** and **DB Reporting** are independently executed, then **DB Updating**, and finally **Consistency Checking**.

The single model constructed by applying the above semantics of environmental elements and generalization-specialization relations is the *fully expanded model*. The woven models can be maintained either as component modules, or as fully expanded models.

3.3. Enhancing basic woven models

Having created the basic woven model, the system architect can continue specifying the combined system in a separate layer without affecting the composing modules.

This layer includes the generalization-specialization relations between the modules and additional intra- and inter-relations. This stage may include reusing aspects, integrating stand-alone components, or modeling new requirements following OPM/Web rules. This stage enables optimization and minimization of the combined modules as a complete application which offers functionality that goes beyond that of each of the individual modules.

4. Reusing OPM/Web models: The Web-based accelerated search example

We use the Accelerated Search System [10] to demonstrate the OPM/Web weaving process described in Section 3. The Accelerated Search System implements an algorithm for improving the performance of a search engine over the Web requiring time-consuming search algorithms. The design of this system includes two models – a reusable one, called **Acceleration**, and a specific one, called **Multi-Search**. The **Acceleration** model specifies a generic algorithm that reduces the execution time of an input-output part of a system by trying first to retrieve the output, which is determined by the input, from a database. If the entry is not already in the database, it activates a process that calculates the sought output and records it in the database to accelerate future executions of the algorithm with the same input. The **Multi-Search** model implements a new search engine that benefits from existing search engines by combining their results and ordering them according to a weighted score. We assume that the needed Web items are static and rarely changed, so the query results remain valid in consequent activations of the query with the same input. Previous query results can therefore be stored to avoid executing the costly search engines.

4.1. The Acceleration and Multi-Search models

Figure 3 specifies that **Acceleration** consists of three main steps: searching the DB for an input-output entry, retrieving the output if it was found in the database, and activating the full process otherwise. The **Acceleration** model contains two environmental objects, **Input** and **Output**, and one simple lowest-level environmental process, **Original Processing**. The scaling rule implies that **Accelerating** and **Full Process Activating** are also environmental processes. All the other things are systemic, as they are internal to the generic model.

Figure 4 is the top-level diagram of the **Multi-Search** algorithm. It specifies the inputs (**Term** and **Query Result Msg**) and outputs (**Query Msg** and **Search Result**) of the algorithm. We could also zoom into **Multi Searching** sub-processes that create queries, send them out, and correlate the results, but these are not needed for the weaving here.

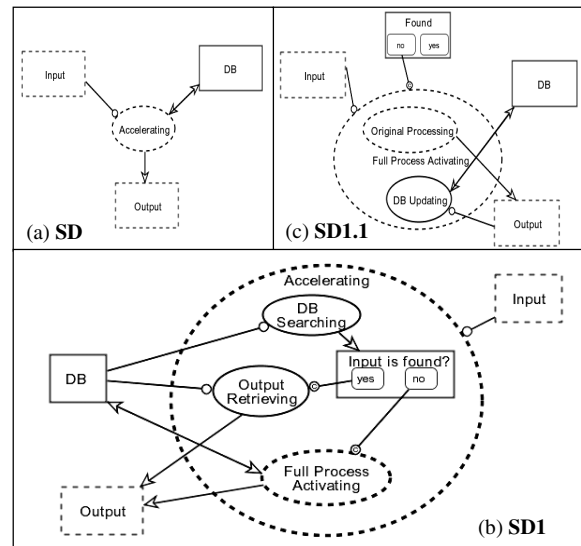


Figure 3. The Acceleration model. (a) Top-level diagram. (b) Accelerating zoomed-in. (c) Full Process Activating zoomed-in.

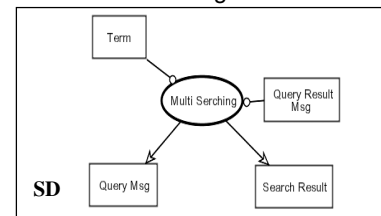


Figure 4. Top level Multi-Search model

4.2. The basic Acceleration/Multi-Search model

To improve the response time, we weave the **Acceleration** model into the **Multi-Search** model, so the latest searched terms and their results are saved in a local database, which is searched before invoking the entire **Multi-Searching** process. Figure 5 shows the basic woven **Acceleration/Multi Search** model with **Acceleration Module** zoomed-into **Full Process Activating**, and the top-level **Multi-Search Module**. Three generalization-specialization relations connect the two modules. The object class **Term** is an **Input** and **Search-Result** is an **Output**. The third generalization-specialization relation is between two process classes, specifying that **Multi-Searching** specializes **Original-Processing**. **Full Process Activating** is implicitly bound to a default systemic process that includes just **Multi-Searching**. These bindings are legal according to the inter- and intra-model weaving rules. If we developed the **Acceleration/Multi-Search** model conventional (by completely specifying the same system without weaving any reusable model), the resulting model would be specific to the problem hindering reuse. The model in Figure 3 enjoys the benefits of generality, which makes it reusable for a variety of functions having the same core architecture.

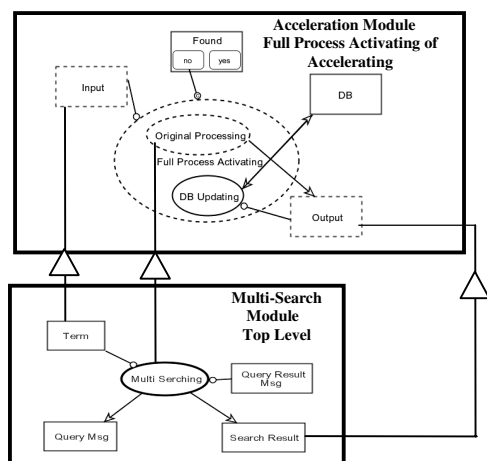


Figure 5. The Acceleration/Multi-Search model

4.3. Further extensions/enhancement

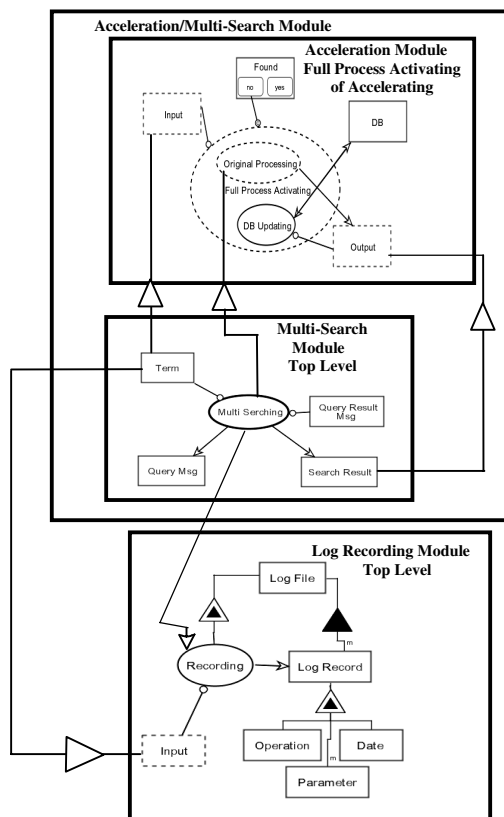


Figure 6. Adding Log Recording

The equivalent semantics of the woven and fully expanded models makes it possible to treat woven models as regular OPM/Web models. The designer can continue specifying the combined system and integrate other reusable models into it. For example, the **Log Recording Module** in Figure 6 includes a **Log File** along with its **Log Records** and a **Recording** operation. When weaving the module into the **Acceleration/Multi-Search Module**, **Term** is bound to **Input**, and the modules are connected with an

invocation link to denote that **Recording** is triggered through a **Multi-Searching** process termination event.

Thus, using the rules and weaving repeatedly, complex new systems can be formed from partially specified components, supporting both open and closed reuse.

References

- [1] The Aspect-Oriented Software Engineering Web site. <http://www.comp.lancs.ac.uk/computing/aop/index.html>
- [2] K.S. Barber, T.J. Graser, and S. R. Jernigan, Increasing Opportunities for Reuse through Tool and Methodology Support for Enterprise-wide Requirements Reuse and Evolution. 1st Intl. Conf. on Enterprise Information Systems, 1999, pp. 383-390.
- [3] G. Booch, Object-Oriented Analysis and Design with Application. Benjamin/Cummings, 1994.
- [4] K. Czarnecki, U. W. Eisenecker, and P. Steyaert, Beyond Object: Generative Programming. AOP -8.
- [5] S. Clarke, and R.J. Walker, Composition Patterns: An Approach to Designing Reusable Aspects. Intl. Conf. on Software Engineering, 2001.
- [6] D. Dori, Object-Process Analysis: Maintaining the Balance between System Structure and Behavior. Jour. of Logic and Computation, 5, 1995, pp. 227-249.
- [7] D. Dori, Object-Process Methodology - A Holistic Systems Paradigm. Springer Verlag, Heidelberg, NY, in press, 2002.
- [8] Components with UML The Catalysis Approach. Addison-Wesley, 1998.
- [9] S. Eckstein, P. Ahlbrecht, and K. Neumann, Increasing Reusability in Information Systems Development by Applying Generic Methods. 13th CAiSE, LNCS 2068, 2001, pp. 251-266.
- [10] Y. Firstenberg, S. Katz, and O. Shmueli, An Object-Oriented Program Accelerator Using Impersonation, Technion Computer Science Department, Technical Report CS-2002-06, 2002.
- [11] W. Frakes, and C. Terry, Software Reuse: Metrics and Models. ACM Comp. Surveys, 28, 1996, pp. 415-435.
- [12] S. Katz, A Superimposition Control Construct for Distributed Systems. ACM TOPLAS, 15, 1993, pp. 337-356.
- [13] N. G. Lester, F. G. Wilkie, and D. W. Bustard, Applying UML Extensions to Facilitate Software Reuse. The Unified Modeling Language - Beyond the Notation. LNCS 1618, 1998, pp. 393-405.
- [14] H. Mili, F. Mili, and A. Mili, Reusing Software: Issues and Research Directions. IEEE Transactions on Software Engineering, 21, 1995, pp. 528-562.
- [15] I. Reinhartz-Berger, D. Dori, and S. Katz, OPM/Web Object-Process Methodology for Developing Web Applications. Annals on Software Engineering: OO Web-based Software Engineering, 2002 (to appear).
- [16] UML 1.3, <http://www.rational.com/media/uml/resources/documentation/ad99-06-08-ps.zip>.