OOSA'98
ECOOP'98 Workshop on Object-Oriented Software Architectures
Workshop Summary

Jan Bosch

University of Karlskrona/Ronneby
Department of Computer Science
SoftCenter, S-372 25, Ronneby, Sweden
E-mail: Jan.Bosch@ide.hk-r.se
WWW: http://www.ide.hk-r.se/~bosch

Helene Bachatene

Alcatel Alsthom Corporate Research Center
Object Architectures Unit
Alcatel Alsthom Recherche
91460 Marcoussis, France
E-mail: Helene.Bachatene@aar.alcatel-alsthom.fr

Görel Hedin

Lund University
Department of Computer Science
P.O. Box 118, S-221 00, Lund, Sweden
E-mail: Gorel.Hedin@dna.lth.se
WWW: http://www.dna.lth.se/home/GorelHedin

Kai Koskimies

Nokia Research Center
Box 422, FIN-00045 Nokia Group, Finland
E-mail: kai.koskimies@research.nokia.com
WWW: http://www.uta.fi/~koskimie/

Introduction

Recently, one can recognize an increased awareness in the object-oriented research community with respect to the role of software architectures. Examples of this can, among others, be found in object-oriented frameworks and design patterns. In the call for papers for the workshop, the following topics were identified where software architecture, patterns and frameworks should be studied in more detail:

- **Building OO software architectures**: How can design patterns, frameworks and components be included in the traditional object-oriented analysis and design? How are architecture constraints processed to build OO software architectures? Which techniques can help to build an OO architecture that supports reuse and is scalable?
- **Language role in architecture**: What is the role of a particular object-oriented language when designing an architecture?
- **Architecture documentation**: It has been suggested that design patterns can be used to document the design and rationale of an object-oriented architecture. How well does this work in practice?
- **OO architectural styles**: What is the relation between the architectural styles of Garlan and Shaw and OO design?
- **Composition of architectures**: Since object-oriented frameworks are an accepted technology within the software industry, the composition of two or more frameworks in an application occurs increasingly often. The composition may be far from trivial and the problems that may appear as well as solutions should be investigated.

- **Component technologies and architectures**: Components are becoming key elements of reusable software development, and various technologies have emerged to support the specification of components and their use in unpredictable combinations (e.g. Java Beans). What are the relations between component-oriented and framework-oriented architectures? How do different component communication mechanisms affect the architecture?
- **Architecture evaluation**: Even though object-oriented frameworks claim to promote reuse and decrease development cost, there are no techniques available to evaluate a concrete framework to determine whether it supports the functional and, especially, non-functional requirements for an application intended to be built based on the framework.
- **'Domain creep'**: Several examples of frameworks exist that, over time, are applied in domains differing from the originally intended domain. The framework then needs to be adapted to include the new domain as well, either as an integrated framework or as two versions. The problems associated with domain creep need to be identified and solution strategies identified.
- **Experiences**: Object-oriented software architectures are used out in industry and academia. However, often no explicit evaluation is performed and experiences are not captured and described objectively. The experiences, case studies and concrete systems and identify problems and weaknesses, but also successful structures and approaches need to be described.
- **Tool support**: A considerable need for architecture tool support exists. Examples are visualization, editing, design, and documentation of architectures.

The goal of the OOSA'98 workshop was to study the aforementioned topics, define the research area and state-of-the-art in more detail and to identify open problems. The call resulted in 15 papers accepted for the workshop. These papers were divided over 5 sessions. Each session was divided in a presentation part, in which the authors briefly presented their work, and a discussion part where the audience was able to discuss the presented and related ideas with the authors and each other. In the remainder of this workshop summary, the results from each session are presented.

Session 1: Experiences

Presentations in this session discussed practical experiences of developing OO software architectures from various points of view. In the first paper *Klaus Marius Hansen* (University of Aarhus, Denmark) presented experiences related to the architectural restructuring in the Dragon project (discussed also in a regular conference paper). The project was carried out in the Mjolner/Beta environment and used an experimental methodology where a prototype was developed and evolved into a final product. As very little was known about the problem domain in the early phases of the project, a simple traditional MVC architecture was chosen as a starting point. During the project, the architecture evolved and was restructured as knowledge about the problem domain was gained. The final architecture is more complex and uses several independent controllers and has a less emphasized model part, with the benefits of better suitability for component-based technologies, representativeness, and flexibility. It was noted that in this kind of experimental development, tools for

iterative analysis and design are needed, supporting both forward and reverse engineering.

Bilhanan Silverajan (Tampere University of Technology, Finland) told about an experience in using an OO framework (OVOPS) for protocol implementation. It was noted that design patterns are truly helpful, but that one often has to modify or combine patterns presented in the literature. Domain creep was also identified as a problem; to overcome this the core framework was kept as generic as possible. The composition of the framework with other systems and frameworks (e.g. CORBA) was found problematic. An interrupt-driven approach was used to integrate OVOPS with CORBA.

Rob van de Weg (University of Twente, Netherlands) discussed the design of an architecture for a production control system. The approach emphasizes domain knowledge as the basis of high-level architectural decisions. In the developed system, the architecture consists of four main layers: base objects controlling hardware devices, relationship objects representing device relationships, coordination objects coordinating the behavior of hardware devices in the production process, and external layer taking care of the user interface.

In the last talk of the session, *Juha Savolainen* (Helsinki University of Technology, Finland) discussed the use of requirement analysis to capture the necessary common features of a product family. The idea is to classify the requirements as common, partial, and unique (with respect to their relevance to different applications in the family). Common requirements define the core framework. Design patterns are used to describe decisions, and Design Decision Trees are used to organize them.

Session 2: Frameworks and Patterns

This session presented the importance of patterns and frameworks in the construction of software architectures. Starting from the general consideration that frameworks are the reusable architectures used to improve development processes, and patterns the solution constructs recurrent in many architectures or designs, discussions of session 2 articulated around combining the two, and highlighting the limits of the two in designing the software architecture of need.

Markku Hakala et al. (University of Tampere, University of Helsinki, Finland) presented a tool and methodology for building application frameworks using patterns. They argue that since design patterns capture solution elements for design, implementation, and documentation of architectural constructs, they form suitable building blocks for new frameworks and applications, thereby giving a systematic approach to the design and reuse of frameworks. *Markku* presented the related methodology and the tool set FRED, a development environment for framework development and specialization. FRED is a result from collaborations between the above mentioned Finnish universities and several Finnish industrial partners. The approach advocates (i) better generation of implementation code than macro expansion, thanks to the explicit representation of design patterns, thus better traceability between designs and their implementations, and (ii) better aiding of framework users when instantiating any structure in FRED, by support from construction wizards. The approach was compared to related pattern-based architecture models, e.g. cloneable software fragments.

Tamar Richner (University of Berne, Switzerland) argued for the necessity of an architectural description of a framework and then analyzed why design patterns on their own are insufficient for such a description. She claimed that traditional artifacts of domain analysis and object-oriented design can better capture the architecture of a framework by describing the solutions in the problem *context*, and proposed a variety of complementary forms of documentation to address the requirement of an architectural description from a higher level granularity than of design patterns. *Tamar* suggested to view software architecture as the different correlations between the *what* and the *how*, close to Kruchten's 4+1 development views, or to aspect programming which advocates multiple views at the programming language level. From a re-engineering perspective, the expected benefit is a better understanding of software architecture decisions, thus better control of software evolution. *Tamar* suggested typical tools to help program understanding from structure, and outlined her future works on understanding application architecture from dynamic information of program execution.
On evolution of software architectures, *Linda M. Seiter* (Santa Clara University, California) discussed the generic evolution patterns common among many design patterns, and the role of a language in supporting evolution of various software architectures. Class-based models and languages require dynamic behavior to be implemented through explicit delegation, using a technique similar to the state, strategy and visitor design patterns. The flaws resulting from using explicit delegation to achieve dynamic implementation were discussed: (i) the relation between interface and implementation is inadequately captured, (ii) additional virtual invocation is required for implementation, and (iii) the host reference (*this*) is not properly maintained. *Linda* presented an improvement of the modeling and implementation of dynamic behavior, through the use of implicit delegation, and proposed an extension to UML class diagram notation to directly model the relation between an instanciable class interface and its multiple implementations. She also investigated language mechanisms for supporting dynamic implementation inheritance within the domain of static, type safe languages (multi-dispatch, predicate dispatching, reflection).

Session 3: Fundamentals of Architectures

The third session aimed at discussing the conceptual fundamentals of software architecture. The session consisted of three papers. The first paper was presented by *Jorge L. Ortega Arjona* (University College London, U. K.) and discussed the relation between *software structure* and *software architecture*. Many treat these concepts as equivalent, whereas others define software architecture as everything but the structure of a software system. The author's conclusion is that software structure is a part of software architecture and should be designed to keep properties such as stability, composability and geometry.
Robb Nebbe (University of Berne, Switzerland) discusses *semantic structure* as the main characteristic for software architecture. Not so much the level of abstraction, but rather the separation of semantic structure from other information defines an architectural description. The author first distinguishes between domain architectures, describing systems of classes, and situation architectures, describing systems of instances. These two are related through *instantiation* and both are static and exclude

all dynamic information. Finally, the aforementioned notions are discussed in relation to some architectural styles, design patterns and frameworks.
Rakesh Agarwal (Infosys Technologies Ltd., India) presented the O^3ML, a model and methodology for describing object-oriented systems. O^3ML allows the software engineer to define operational models of a software system and to simulate the conceptual, functional and implementation aspects of the model. The author explicitly disthinguishes between persistent and transient parts of software and suggests to explicitly model the persistent parts to guide software evolution.

Session 4: Dynamic and Flexible Architectures

This session contained two papers discussing techniques supporting flexible architectures. *Jesper Andersson* (Linköping University, Sweden) presented a technique for dynamically updating a class. Several ways to implement this kind of facility in Java were discussed. The techniques are based on having an optional reference in a class object to the next version of the class, if available. An object can be replaced by a new version either by modifying the reference load operation (load-time substitution) or the dereference operation (dereference-time substitution) of the virtual machine. The latter is more safe and efficient. In the discussion it was noted that prototype languages might better support this kind of dynamic updating.
Frank Matthijs (University of Leuven, Belgium) presented the idea of reflection points as a means to achieve a dynamically configured architecture. A reflection point is an "intelligent" connector that chooses the destination on the basis of its observations of the environment. The idea is applied to construct a flexible architecture for a protocol stack. For example, in a TCP/IP environment, the IP layer can be made flexible by letting reflection points choose between different kinds of upper or lower layers. Hence the appropriate layer type combination can be dynamically selected. If some choices are predetermined, the corresponding reflection points can be effaced, thus avoiding unnecessary computation.

Session 5: Architectural issues in design

Session 5 contained three papers discussing architectural issues in design. The main issues identified during the session were the importance and limits of architectural patterns, the difficulty to instantiate frameworks designed through examples, and the difficulty to address architecture "ilities" (i.e., reliability, consistency, and other quality issues) using existing patterns.
Sophie Cherki (Thomson-CSF Corporate Research Laboratory, Orsay, France) presented a view on how object-oriented methods and software architecture complement one another, by emphasizing an architecting phase and the way it integrates into a traditional OO development cycle, and the use of OO techniques such as patterns to capture solutions and enforce design by context analogy.
William van der Sterren (Philips Research Laboratories Eindhoven, the Netherlands) summarized the difficulties to issue a system architecture and to address system "ilities" using actual OO techniques. The decomposition of the system in hardware

components and software components is a hard issue as many forces (cost, hardware performance, distribution, standards, legacy) affect the decomposition. However, choosing hardware implementations for some components has architectural consequences. Moreover, the pattern Master-Slave to achieve reliability, is too generic to obtain the needed system uptime. Looking at existing broadcast systems it appears that clustering technology offers a solution to increase the uptime in the presence of hardware, and sometimes, software errors. However, patterns are needed that take into account architectural impact of clustering technology.

Marcelo Campo (UNICEN, Buenos Aires, Argentina) discussed the use of architectural styles to guide the initial steps of framework design, based on two experiences: *luthier*, a framework for analyzing programs, and *baf*, a framework for building multi-agents systems. Three interesting points were issued from these experiences, namely (i) the utility of Garlan & Shaw architectural styles, (ii) the importance of design patterns in the development of frameworks, and (iii) the difficulty of instantiation of example-driven frameworks. The later problem could be avoided if development is based on a pre-defined architecture, as confirmed by experimental developments of a framework based systems.