# Software Architecture Recovery and Modelling

## [WCRE 2001 Discussion Forum Report]

Arie van Deursen
CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands
http://www.cwi.nl/~arie/

## ABSTRACT

This paper covers current trends and issues in software architecture recovery. It consists of a summary of the presentations and discussions of the *Software Architecture Recovery and Modelling* discussion forum held during WCRE 2001, the *Working Conference on Reverse Engineering*, Stuttgart, Germany, October 2, 2001.

## Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architecture; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Reverse Engineering*

## General Terms

Architecture reconstruction, reengineering, system understanding

## 1. INTRODUCTION

Software architecture recovery aims at presenting existing software systems at the more abstract, architectural level. The Software Architecture Recovery and Modelling (SWARM) discussion forum, a satellite event of the IEEE Working Conference on Reverse Engineering 2001 (WCRE), was intended to give researchers in the area of architecture extraction an opportunity to discuss their progress and explore future directions. The goals of the forum were

- to exchange experience;
- to discuss new application areas;
- to discover areas of mutual collaboration; and
- to envision future trends in the field of software architecture recovery.

This report summarizes the key results of the SWARM discussion forum.

The schedule of the forum is listed in Table 1. The forum was attended by 30 participants. The format included plenty of space for discussion, both at the start and the end of the forum, as well as before and after each presentation and session. In order to stimulate informed debate, each position paper had a discussant assigned, who had the task of studying the position paper in advance, preparing one or two questions. This report covers the topics raised in the presentations as well as the key issues addressed in the discussions.

## 2. TOOL SUPPORT

Claudio Riva and Yang Yaojin outlined an architecture reconstruction process and the environment supporting it, based on experience described in [7]. The process consists of four steps, which are applied iteratively in order to extract a more and more focused view of a system's architecture. The four steps are:

1. Definition of architecturally significant concepts.
2. Data gathering, in which a model of a system is built in terms of the concepts defined in step 1.
3. Abstraction, in which the model is enriched with (domain-specific) abstractions that lead to a higher view of the system.
4. Presentation of the reconstructed architecture in a series of formats, such as graphs, hyperlinks, UML diagrams, and message sequence charts, taking the required architectural view (logical, process, physical, development [5]) into account.

The extraction of the source code model is supported by source code analyzers. Abstractions are computed using Prolog rules, Presentation is done using Rigi visualization [9] enriched with HTML hyperlinks.

One of the issues pointed out in the discussion was that this process can capture the *structure* resulting from a development effort; what it does not cover is the *intent* of the designer. Once a developer leaves a company, his architecture disappears.

## 3. ARCHITECTURE DESCRIPTIONS

Amnon Eden's paper addresses the way in which an extracted architecture should be represented. A language for specifying an architecture should ideally be expressive, well-defined, abstract, concise, compact (small), permitting formal reasoning, and supported by tools.

**Session I: Reconstruction**

Chair: Peggy Storey

| | |
|---|---|
| 14:15 | Welcome |
| 14:20 | Introduction |
| 14:30 | *An Environment for Architecture Reconstruction*<br>Claudio Riva and Yang Yaojin (Nokia Research Center) |
| 14:45 | *Directions in Architectural Specifications*<br>Amnon H. Eden (Concordia University), presented by Claudio Riva. |
| 15:00 | *A Lightweight Architecture Recovery Process*<br>Davor Svetinovic and Michael Godfrey (University of Waterloo) |
| 15:15 | *Recovering Rationale*<br>Arie van Deursen (CWI) |
| 15:30 | Discussion |
| 15:45 | Tea break |

**Session II: Components**

Chair: Mike Godfrey

| | |
|---|---|
| 16:15 | *Towards an Architecture for Refactoring Embedded Software for Ubiquitous Environments*<br>Jens Jahnke (University of Victoria) |
| 16:30 | *Architecture Recovery for Distributed Systems*<br>Nabor C. Mendonça (Universidade de Fortaleza), presented by Nicolas Anquetil |
| 16:45 | *Issues in Reengineering the Architecture of Component-Based Software*<br>J. M. Favre, H. Cervants (Imag), R. Sanlaville, F. Duclos, and J. Estublier (Dassault Systems) |
| 17:00 | *A challenge of modeling how to use an architecture*<br>Tarja Systä (Tampere University of Technology) |
| 17:15 | Discussion |
| 17:30 | Wrap up |
| 17:45 | Closing and Dinner |

**Table 1: SWARM Schedule**

This gave raise to a discussion on Architecture Definition Languages. Most ADLs are formally defined, but their actual use in industry is fairly limited. A further question of interest is whether formality is of importance to architecture extraction. In safety critical areas this may be the case. An architecture however, is an abstraction, and represents, to a certain extent, always a lie (although it should be a "useful lie"). As an example, a system may make use of the mediator pattern, but violate some of the pattern's structure. In such a case, it can still be useful and meaningful to explain the system in terms of the mediator pattern.

## 4. PROCESS

Davor Svetinovic and Mike Godfrey proposed a light-weight architecture recovery process. Their aim is to connect architecture recovery to highly iterative and development processes (such as extreme programming and other agile processes). A characteristic of such processes is that they involve extensive architecture refactorings.

The architecture recovery process proposal is based on experience in using the PBS system for analyzing software evolution in open source systems such as Linux [1]. The main activities include

- Derivation of initial goals and a conceptual architecture;
- Extraction of the code level structure;
- Iterative goal-responsibility code instrumentation;
- Documentation of the architecture and its traceability relation;
- Representation of the concrete architecture and its rationale.

## 5. RECOVERING RATIONALE

Arie van Deursen covered the problems involved in recovering the *rationale* of architectural decisions, based on experience in the redocumentation of large Cobol systems [2]. Recovering rationale is important as well as difficult. The best we can do is:

- Use existing documentation, comments, and log messages as rationale pointers: System browsers aiming at architecture presentation can integrate these with other views;
- Recognize design patterns, which aim at capturing specific rationale knowledge of recurring solutions. For procedural systems, design pattern recovery is complicated by the lack of an explicit pattern catolog and architectural drift of legacy systems;
- Record rationale by combining architecture browsers with either simple annotation mechanisms or more involved rationale capturing tools.

In the discussion it was brought up that many companies, such as Nokia, do maintain large project databases, containing records of discussions, change requests, and inspection results. The challenge is how to organize such a wealth of information to extract just the rationale for architectural decisions. "Decision trees" may be suitable for explaining such decisions. This may involve a certain amount of "post-hoc rationalization" as well.

## 6. UBIQUITOUS SYSTEMS

Jens Jahnke addressed issues concerning embedded, ubiquitous software systems, based on his work on net-centric embedded applications [4]. Dynamic, ubiquitous systems require small components that can be easily glued together in many different ways. Existing embedded software systems, however, are typically not very well componentized, due to strong optimization requirements and platform dependencies.

Migrating such embedded systems to a "connection-based" infrastructure would make them amendable to inclusion in ubiquitous environments. The component refactoring process envisioned to do this consists of four steps:

- Identifying the individual components;
- Reverse engineering the component interfaces;
- Transformation of the component interfaces;
- Slicing components out of the system.

In the subsequent discussion the notion of "architecture evolution languages" was brought up. These should help in evolving environments, to describe constraints and the overall system topology. Another issue that was raised is what role UML could play in such a refactoring process. UML is essential for communicating with end users, as it is visual and well known. One way of using UML is through its *stereotype* extension mechanism, creating specific boxes and arrows that can be useful for architecture modelling. In the area of embedded systems, however, SDL is a useful alternative, as it is more signal-oriented.

## 7. DISTRIBUTED SYSTEMS

Nabor Mendonça's paper reports on the use of the X-ray approach to recovering architectures of distributed systems [6]. One of the current difficulties in architecture extraction is the recovery of runtime abstractions, such as clients, servers, and interaction protocols, which are typical to distributed systems. X-ray aims at finding such dependencies through by combining various *static* analysis methods, thus avoiding the difficulties of performing a dynamic, runtime analysis, which can be expensive, hard to link to the source, and which may affect real time constraints.

X-ray aims identifying implemented executable components and their potential runtime interconnections. X-ray consists of

- Component module classification, mapping sources to executables;
- Syntactic pattern matching, finding runtime interaction features;
- Structural reachability analysis, mapping interaction features to individual components.

A typical example is analyzing socket communication in a software system: this involves finding runtime dependencies, as opposed to compile time dependencies that can be found by analyzing, for example, a Makefile.

## 8. COMPONENTS

Jean-Marie Favre addressed various issues in reengineering architectures for component-based software systems, based on his work on the Generic Software Exploration Environment [3]. A first concern is that the words "architecture" and "component" have different meanings in various communities:

- In the *object-oriented* community, architectures are expressed in terms of *patterns* and *frameworks*.
- In the *ADL* (architecture description languages) community, several languages have been proposed that describe architectures in terms of connectors, constraints, behaviors, and so on. While this approach provides an excellent conceptual framework, until now ADLs have had almost no impact on software practice.
- In the *reengineering* community, a strong emphasis on the *structural* aspects of architecture exists, while the behavioral issues as covered by ADLs are often neglected.
- In the *component-based software engineering* community, the focus is on industrial strength component models, such as Corba, .NET, and EJB.

Architecture recovery research should try to integrate these communities, rather than focussing on one of them exclusively.

## 9. USING ARCHITECTURES

Tarja Systä took up the challenge of modelling how to *use* an architecture. For software products, the *user's manual* serves this purpose. Likewise, for existing software systems, maintainers and reengineers may want to know how to (re)use the services provided by the architecture, or the software components offered by it. As done in user's manuals, example use cases and scenarios, possibly including code fragments, can be used for this purpose.

Traditionally, the focus of architecture recovery and reverse engineering has been on structural models that can be analysed for finding answers to questions starting with *What* [8]. When considering the problem from the maintainer's or reengineer's point of view, we should pay attention to constructing models that give answers to questions starting with *How*. This way of thinking raises new questions and problems concerning information extraction as well as information visualization and documentation.

## 10. SUMMARY

SWARM consisted of 3.5 hours packed with discussion and exchange of ideas in the area of software architecture recovery. The main results from the forum include:

- Software architecture extraction is an iterative, tool-supported process;
- ADLs and UML can be used to describe certain elements of derived architectures, but their application in architecture extraction modelling has been limited so far.

- Architecture recovery research has been successful in the area of extracting *structure*; recovering design decisions and rationale is significantly harder, if not impossible.
- Embedded, ubiquitous, and distributed software systems have special characteristics requiring specific approaches to software architecture recovery.
- Successful architecture recovery research should integrate results from the object-oriented, ADL, reengineering, and component-based communities.
- Viewing architecture from the *user's* (maintainers and reengineers) point of view raises new recovery research areas.

The full versions of the papers are available from the SWARM web site, at `http://www.cwi.nl/~arie/swarm2001/`.

## Acknowledgments

A warm thanks to the 30 enthusiastic participants. Special thanks to the four co-organizers: Mike Godfrey (University of Waterloo), Rainer Koschke (University of Stuttgart), Claudio Riva (Nokia Research Center), and Margaret-Anne Storey (University of Victoria).

## 11. REFERENCES

[1] I. T. Bowman, R. C. Holt, and N. V. Brewster. Linux as a case study: Its extracted software architecture. In *21st International Conference on Software Engineering, ICSE-99*, pages 555–563. ACM, 1999.

[2] A. van Deursen and T. Kuipers. Building documentation generators. In *International Conference on Software Maintenance, ICSM'99*, pages 40–49. IEEE Computer Society, 1999.

[3] J.-M. Favre. $G^{SEE}$: a generic software exploration environment. In *Proceedings International Workshop on Program Comprehension (IWPC)*. IEEE Computer Society, 2001.

[4] J. Jahnke. Engineering component-based net-centric systems for embedded applications. In *Proceedings European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC, FSE-9)*, pages 218–227. ACM Press, 2001.

[5] P. B. Kruchten. The 4 + 1 view model of architecture. *IEEE Software*, pages 42–50, November 1995.

[6] N. C. Mendonça and J. Kramer. An approach for recovering distributed system architectures. *Automated Software Engineering*, 8(3/4):311–354, 2001.

[7] C. Riva. Reverse architecting: An industrial experience report. In *Proceedings 7th Working Conference on Reverse Engineering (WCRE)*, pages 42–51. IEEE Computer Society, 2000.

[8] T. Systä, K. Koskimies, and H. Müller. Shimba – an environment for reverse engineering Java software systems. *Software Practice & Experience*, 31(4):371–394, 2001.

[9] K. Wong, S.R. Tilley, H.A. Müller, and M.-A.D. Storey. Structural redocumentation: a case study. *IEEE Software*, 12(1):46–54, 1995.