

An Environment for Architecture Reconstruction

Claudio Riva and Yang Yaojin

Nokia Research Center

P.O. Box 407, FIN-00045, NOKIA GROUP

{ claudio.riva | yang.yaojin }@nokia.com

ABSTRACT

A software architecture description should communicate the essential design decisions taken during the design of a system. Architecture reconstruction concerns with the task of recovering past decisions for an existing system. We outline our architecture reconstruction process and the environment supporting it.

Keywords

Reverse engineering, architecture reconstruction, software architecture, tools, XML.

1 INTRODUCTION

A description of a software architecture should communicate the essential decisions that have been taken in the design of a software system. Ran [2] points out that the essential decisions of a design are the ones that are expensive to change and, therefore, the most critical for the development and maintenance of a system. There are four categories of design decisions: *concepts*, *architecturally significant requirements*, *structure* and *texture*. Concepts concern the way we think of a system (an operating systems may use the concepts of tasks, processes, queues, etc.). Decisions about the system concepts are probably the most important ones and can be hardly changed in the later stages of the development. Architecturally significant requirements are the major concerns that should be addressed by a proper software architecture. They should focus on the critical properties that we want to achieve in a system. The structure describes the decomposition of the system in interdependent components and their relationships at the right level of abstraction. Texture is about the design decisions that are taken at the implementation level and are architecturally relevant (design patterns, policies). According to [2], we define software architecture as “*a set of concepts and design decisions about structure and texture of software that must be made prior to concurrent engineering to enable effective satisfaction of architecturally significant, explicit functional and quality requirements, and implicit requirements of the problem and the solution domains*”.

Architecture reconstruction (or reverse architecting) concerns with the task of recovering the past design decisions that have been taken during the development of a system. They comprise either decisions that have been lost (because not documented or developers have left) or are unknown (for examples, assumptions not initially take into account). The goal is to put light in all the categories of design decisions that are relevant for a software architecture description. The reconstruction is performed by examining the available artefacts (documentation, source code, experts) and by inferring new architectural information that is not immediately evident. Our approach can be summarised in the following four-step iterative process [3]:

1. Definition of architectural concepts

The goal of this phase is to recover and clarify the architecturally significant concepts that build the system. These concepts represent the way developers think of a system and they should become the terminology of the reconstruction process. They concern the building blocks of the system and the communication infrastructure that enables the components to communicate at runtime. These concepts are usually evident in the reference architecture document otherwise they have to be reverse engineered. This step has also to identify how the architectural concepts are mapped to the implementation. In a distributed software system the architectural concepts may be applications, servers, software busses while in an operating system they may be tasks, processes, queues, shared memories, etc. Textures should also be considered at this stage. For example, a design pattern may hide an interaction pattern that is architecturally significant.

2. Data gathering

This phase gathers the relevant information for describing the software architecture of the system. We build a model of the system whose entities are instances of the concepts identified in the phase 1. A correct choice of the concepts will ensure that the model is filled with entities at the right level of abstraction. This phase is mainly concerned with data gathering more than with reasoning on the architecture. Therefore, this task can be easily automated with tools.

Different sources of information are involved in the process. Source code is usually the most dependable for static analysis [3] and simulation for dynamic analysis. However, documentation, software diagrams (for example, stored in CASE tools), experts can contribute to the creation of the model.

3. Abstraction

The model of the previous phase is usually at a very low level of abstraction. The goal of this phase is to enrich the model with domain dependent abstractions that will lead to a high level view of the system (that is the structure of the architecture description). Known abstractions can be easily added to the model. Unknown abstractions have to be identified by the architects, categorised, named and then stored in the model. Such reasoning is conducted manually by the architects and then fixed in a set of abstraction rules. The abstraction process has also to produce the architectural views that will be presented in the last phase (for example, according to the 4+1 model [1]).

4. Presentation

The architects need to present the reconstructed architecture in different formats. We allow the architects to select a particular architectural view (logical, process, physical, development) and a particular format: graphs (using Rigi), web format (with hyperlinks), UML diagrams and message sequence charts.

2 THE ENVIRONMENT

We outline an environment for architecture reconstruction that we are currently using and extending. The environment integrates several tools to support three main tasks: model capture, abstraction and presentation. We use GXL (Graph eXchange Language) for data exchange. The tools are integrated using scripting languages like Perl and Tcl/Tk. XSLT scripts are used for converting the GXL files to different formats. Prolog is used for the abstraction steps.

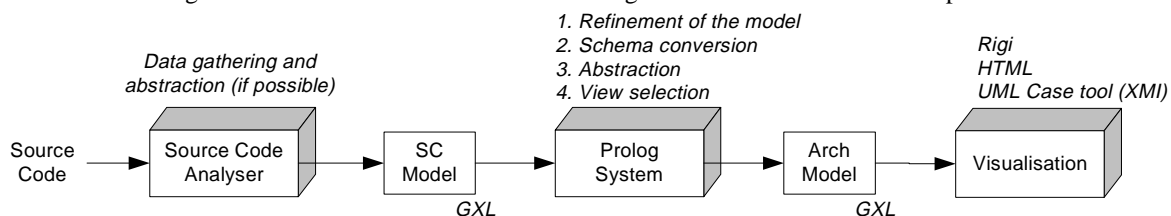


Figure 1. Reverse architecting steps.

Model capture

The extraction of the source code model is achieved by source code analysers: ad hoc analysers written in Perl scripts or converters from the symbol tables of programming environments like SourceNavigator and Sniff+. The output is a set of relational data representable with a directed typed graph and stored in GXL. Relying on external analysers enable us to support a variety of languages in a totally independent way.

Abstraction

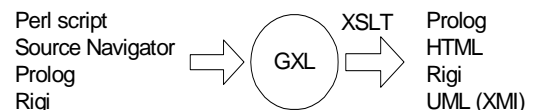
The phase of abstraction is supported by a Prolog system. We use Prolog to formally specify the following operations: refinement of the model (mistakes are removed and additional facts are added), schema conversion (facts are manipulated in order to infer new relationships), abstraction (facts about the grouping of the elements and the abstractions are added) and view selection (the model is projected in several views that can be visualised). The facts about the abstraction rules are provided by separate files that are manually prepared by the architect.

Presentation

We can present the views of the architectural model with oriented typed graphs using Rigi and a set of web pages in HTML format with hyperlinks. We also plan to generate diagrams in UML notations by converting the GXL file in XMI format.

Format conversions

The models are stored in the GXL format, which plays the role of a relational data repository. The conversion of the data to other formats is achieved with XSLT scripts. XSLT allows us to select parts of the GXL file and print them in RSF format, Prolog facts and HTML (in the future XMI too). We produce GXL files from the source code analysers, from Prolog and Rigi.



REFERENCES

1. Kruchten P.B., The 4+1 View Model of architecture, IEEE Software, 12(6):42-50, 1995.
2. Ran A., "ARES Conceptual Framework for Software Architecture" in M. Jazayeri, A. Ran, F. van der Linden (eds.), *Software Architecture for Product Families Principles and Practice*, Addison Wesley, 2000.
3. Riva C., Reverse Architecting: an Industrial Experience Report, *Proceedings of the 7th Working Conference on Reverse Engineering (WCRE2000)*, Brisbane, Australia, 23-25 November, 2000.