Software architecture reconstruction

Krikhaar, R.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 2

Overview of the SAR Method

In the previous chapter we gave an overview of software architecture. In this chapter we present a framework required for a method to reconstruct a software architecture of an existing system.

2.1 Introduction

Here, we introduce a method to reconstruct an existing system’s software architecture: the Software Architecture Reconstruction (SAR) method. We discuss a general framework for the SAR method, which is also used to structure this thesis.

In general, all methods consist of four different parts [Kro93]:

- an underlying model;
- a language;
- defined steps and ordering of these steps;
- guidance for applying the method.

In our software architecture reconstruction method, the underlying model consists mainly of Relation Partition Algebra (to be elaborated in Chap-
ter 3). Relation Partition Algebra consists of sets, binary relations, part-of relations and operations on them. Besides a model, RPA is also a language for expressing architectural information: we need graphical and textual notations (graph diagrams, relation tables, lists) to present a reconstructed software architecture.

The reconstruction of software architecture consists of performing the following kinds of steps: extraction, abstraction and presentation (see Section 2.3). Extraction steps will be discussed as parts of Info Packs (the notion of an InfoPack will be discussed in Section 2.5.2); abstraction and presentation steps are contained in ArchiSpects (the notion of an ArchiSpect will be discussed in Section 2.5.2). A guidance describes the gaps that are not completely covered by the steps or when the steps do not perfectly fit in the situation at hand.

In this chapter we briefly describe the engineering of software architectures (called forward software architecting). Next, we will discuss reverse software architecting, which is the counterpart of forward software architecting. As we will see, improvements in existing software architectures demands both engineering disciplines. We will finish this chapter with a framework into which the software architecture reconstruction method can be fitted.

### 2.2 Forward Software Architecting

Forward software architecting, or simply software architecting, is the discipline of engineering a software architecture from scratch, or, if an architecture already exists, it consists in engineering the extensions of the architecture. An example of a method dedicated to architecture is the Building Block method [KW94, LM95].

In chapter 1 we have discussed a number of architectural patterns that are related to the module view of software architecture. One can also define architecting as the process of selecting and applying proper patterns for each of the architectural views. It is an engineering discipline that requires a lot of experience, human sense, knowledge of a range of good architectural patterns and the ability to define new appropriate architectural patterns.
Reverse software architecting is the flavour of reverse engineering that concerns all activities for making existing (software) architectures explicit [Kri97]. Reverse software architecting aims for: recovery of lost architectural information, updating of architecture documentation, supporting of maintenance (comprehension) activities, provision of different (other) views on architecture, preparing for another platform and facilitating impact analysis. Reverse engineering was defined as follows by Chikofsky and Cross [CC90]:

"The process of analysing a subject system to identify the system’s components and their relationships and create representations of the system in another form or at a higher level of abstraction."

Figure 2.1 (taken from [CC90]) presents a lot of terminology within a simplified software life-cycle. Requirements involves the specification of the problem, design is the specification of a solution and implementation concerns the creation of a solution which consists of coding, testing and system delivery. Redocumentation is the simplest and oldest form of reverse engineering. It concerns the creation or revision of a system’s documentation. However, many tools dedicated to redocumentation are only able to generate diagrams, print code in an attractive way, or generate cross-reference
Restructuring is the transformation from one representation form into another, preserving the external behaviour. The first experiments in this area concerned the removal of 'goto' statements and their replacement by control structures like 'while' loops, 'if-then-else' clauses and 'for' loops. Design recovery means that one identifies meaningful higher levels of abstraction of software. For this activity one requires domain knowledge and designer's knowledge to add the information required to be able to create these abstractions. Reengineering is related to the modification of an original system to increase design quality.

### Extract – Abstract – Present

The process of reverse engineering (depicted in Figure 2.2) in general consists of three activities:

- **extract**: extracting relevant information from system software, system experts and system history;
- **abstract**: abstracting extracted information to a higher (design) level;
- **present**: presenting abstracted information in a developer-friendly way, taking into account his or her current topic of interest.

Tools can be used to extract information from the system software, which includes source code, design documentation, etc. The value of the tool's output may depend on the availability of coding standards, and of course
on whether these coding standards are satisfied by the developers. For example, many implementation languages do not explicitly support a module concept (similar to modules in Modula-2), but one can force a pseudo-module concept by prescribing certain coding rules. The extraction results are stored in a database, which is called a repository. System experts can be interviewed to obtain architectural information with the aid of different techniques, e.g. think-aloud sessions, structured interviews and brain-dump sessions. History information can be extracted from the software archive or documentation system, providing information about the system’s evolution.

Because most of the extracted information is often at programming level, one must abstract from this information and bring it to an architecture level. In addition, some filtering of information may be required for certain views on the system. Developers need different views on (parts of) the system in their daily work. The requested view is to a great extent driven by the problem at hand, so good navigation means are needed to retrieve information.

The abstracted information can be presented in different ways. Developers may prefer diagrams and pictures, but more fancy media may be applicable such as sound and vision, instead of textual descriptions, e.g. lists of items. Hyperlinks should be added to textual descriptions to achieve good navigation means. All these types of presentation types have already been integrated in various Web browsers, which makes this medium a good candidate for these purposes.

Extracted information may originate from different tools, e.g. if multiple implementation languages are used. Combining this information from different sources may result in incomplete or even conflicting data. One must allow for such situations, noting that incomplete data may appear complete at higher levels of abstraction.

In the appendices we give an overview of the tools that proved to be useful during our research. Extraction tools are discussed in Appendix A. Abstraction tools based on Relation Partition Algebra are presented in Appendix B. In Appendix C we discuss some proprietary presentation tools.
2.4 Architecture Improvement

One can improve a system by starting from scratch again and rebuilding the complete system. However, this is hardly an option for systems containing software of hundreds of person-years’ development. Another approach starts with the existing system as a basis and incrementally improves (parts of) the system.

Figure 2.3 (taken from [Kri97]) shows a process for the latter approach, comprising three typical activities:

- **forward architecting** uses architectural objectives and functional requirements as input for the definition of an ideal architecture.
- **reverse architecting** consists of creating explicit architectural models of an existing system, the as-built architecture. Traditional reverse engineering techniques can be applied to extract information from software artefacts. Appropriate abstractions must be made to obtain the information at an architectural level.
- **re-architecting** involves balancing an ideal architecture against the existing architecture to prioritize a list of desired improvements. The
next step is to implement a number of these improvements. The size of the improvement steps depends on both business-related issues and technological facts.

Besides software architecting experience (including knowledge of architectural patterns), a lot of domain and system knowledge is required to define an ideal architecture. Reverse architecting can help in extracting domain knowledge from the system, but it can also clarify existing architectural patterns in the system. The recovered patterns may influence choices made during forward architecting. Of course, badly chosen patterns should not be copied by the new ideal architecture, but should be seen as cautions. The current system can give clues for defining a new architecture in a positive sense by recovering existing patterns. But it can also show the design decisions that failed in the past, which must be avoided in the new system.

The whole process is iterative, in the sense that after improvements have been implemented reverse architecting can help to make explicit the created architecture, which may differ from the initially defined architecture. This process is in fact similar to any improvement activity: define the current situation (or check the previously realized improvement), define the desired situation, and define the path to reach the desired situation and execute it.

2.5 The SAR Method

Relation Partition Algebra, architectural views, reconstruction levels, InfoPacks and ArchiSpects are the key elements of our Software Architecture Reconstruction (SAR) method. Architectural views have already been discussed in Section 1.2.3 and RPA will be presented in Chapter 3. Before we can focus on the details of our SAR method, we have to present the notion of reconstruction levels, InfoPacks, ArchiSpects, and a framework in to which these notions can be fitted.

2.5.1 Software Architecture Reconstruction Levels

We introduce different levels\(^1\) of software architecture reconstruction. Each SAR level covers a range of architectural aspects that must be reconstructed.

---

\(^1\)We have been inspired by the levels in the Capability Maturity Model [Hum89].
Consider a system which is hardly documented and whose software architecture is not known. Such systems are at the initial level of reconstruction. By making the software architecture of such a system explicit (i.e. reverse architec
ting the system), we reach the described level of SAR\(^2\). If the gap between the ideal software architecture and the described software architecture is too big, one must improve it by redefining parts of it. Then, by re-architecting the system, we reach the redefined SAR level. After the architecture improvement, one must sustain the reached quality level. Without any precautions, the architecture will certainly degenerate after a while. If we can continuously preserve the software architecture in a controlled way, we reach the managed level. Now that we have the software architecture completely under control, we can optimise the architecture for all kinds of future extensions, which is called the optimised level. So, the following software architecture reconstruction levels exist:

- initial level;
- described level;
- redefined level;
- managed level;
- optimised level.

### 2.5.2 InfoPack and ArchiSpect

We introduce the terms InfoPack and ArchiSpect as the components of our software architecture reconstruction method. An InfoPack\(^3\) is a package of particular information extracted from the source code, design documents or any other information source. An InfoPack contains a description of the extraction steps to be taken to retrieve certain software information. Alternative extraction techniques may exist for different programming languages, which are discussed as parts of the InfoPack. Sometimes an InfoPack is specific to a certain (programming) language or class of languages, e.g. InfoPacks working with the notion of inheritance are of interest only for object-oriented languages. InfoPacks may also be domain- or application-dependent, which makes them less widely applicable. Examples are the Import InfoPack, import dependency extraction, and the Part-Of InfoPack, extraction of the decomposition hierarchy. An ArchiSpect\(^4\) is a

---

\(^2\)We may assume that each system (especially systems that exist for many years) contains some notion of software architecture, although it has not been explicitly documented.

\(^3\)The term InfoPack is an abbreviation of the phrase information package

\(^4\)The term ArchiSpect combines the words architecture and aspect.
view on the system that makes explicit a certain architectural structure. An ArchiSpect is more abstract than an InfoPack and therefore more widely applicable. Most ArchiSpects build upon the results of Info Packs. A complete set of ArchiSpects in fact describe a system’s actual architecture; the Info Packs serve as supporting units to construct the ArchiSpects. Besides abstraction of information, an ArchiSpect covers possible ways of presenting architectural information. Examples are the Component Dependency ArchiSpect, recovery of dependency between the components of a system, and the Layering Conformance ArchiSpect, verifying whether a system is correctly layered.

In this thesis we describe Info Packs and ArchiSpects according to a fixed scheme:

- **Name**: the name of the Info Pack or ArchiSpect;
- **Context**: the architectural view (see Section 1.2.3) to which it belongs and the related Info Packs and ArchiSpects (as will be clarified in the description);
- **Description**: an introduction to the Info Pack and ArchiSpect;
- **Example**: typical example(s) from Philips’ systems (as appeared after the method had been applied);
- **Method**: a description of the steps that must be taken to construct the Info Pack or ArchiSpect;
- **Discussion**: discussion of items not addressed in one of the above sections (e.g., discussion of related work).

We can look at Info Packs and ArchiSpects in different ways. The method view focuses on the description of steps and guidance. The tool view concerns the tools required to support the application of ArchiSpects and Info Packs. The representation view contains the results of ArchiSpects and Info Packs. We will use the terms ArchiSpect and Info Pack for each of these views; the context in which the term is used will clarify its actual meaning.

### Software Architecture Reconstruction Framework

For each of the SAR levels (besides the initial level\(^5\)), we can fill out a matrix as given in Table 2.1; the rows contain the various SAR levels and the columns contain the architectural views. The cells have been filled with Info Packs and ArchiSpects. Info Packs are closely related to dedicated ex-

\(^5\)The initial level refers to the situation that no reconstruction has taken place, so the cells are empty.
<table>
<thead>
<tr>
<th>SAR levels</th>
<th>Architecture Views</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logical View</td>
</tr>
<tr>
<td>Optimized</td>
<td>. . .</td>
</tr>
<tr>
<td>Managed</td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td>Redefined</td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td>Described</td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td>Initial</td>
<td>. . .</td>
</tr>
</tbody>
</table>

Table 2.1: Software Architecture Reconstruction Framework

traction means and therefore they appear in the code view column. All architectural views are in fact important, but in this thesis we will concentrate on the module view and code view (the non-dotted area in the SAR matrix). In Chapter 7 we will fill out the SAR matrix with the names of discussed Info Packs and ArchiSpects as we experienced to be useful for the various SAR levels.