

Moving Towards Quality Attribute Driven Software Architecture Reconstruction

Christoph Stoermer

Robert Bosch Corporation
4500 Fifth Avenue
Pittsburgh, PA 15213 USA
+1 412 268 3949
cstoerme@sei.cmu.edu

Liam O'Brien

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213 USA
+1 412 268 7727
lob@sei.cmu.edu

Chris Verhoef

Free University of Amsterdam
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
+31 20 4447760
x@cs.vu.nl

ABSTRACT

There are many good reasons why organizations should perform software architecture reconstructions. However, few organizations are willing to pay for the effort. Software architecture reconstruction must be viewed not as an effort on its own but as a contribution in a broader technical context, such as the streamlining of products into a product line or the modernization of systems that hit their architectural borders. In these contexts software architects frequently need to reason about existing systems, for example to lower adoption and technical barriers for new technology approaches. We propose a Quality Attribute Driven Software Architecture Reconstruction (QADSAR) approach where this kind of reasoning is driven by the analysis of quality attribute scenarios.

This paper introduces a quality attribute driven perspective on software architecture reconstruction. It presents a technical reasoning framework and illuminates the information that is required from the reconstruction process to link the knowledge gained back to the business goals of an organization. The paper illustrates the techniques by presenting a real-world case study.

Keywords

Architecture, Architecture Reconstruction, Architecture Views, Product Lines, Quality Attributes, Quality Attribute Driven Analysis.

1 INTRODUCTION

Much research has been done in software architecture reconstruction (SAR) in the past several years [5,11,16,17,18,21,23,25,27,31,32] and many techniques and methods have been developed along with tools to support them [10,15,22,29]. But why are only few organizations carrying out architecture reconstructions?

There should be many reasons for organizations to perform software architecture reconstructions. For example:

- Re-documenting the architecture of existing systems

- Checking the conformance of an as-designed architecture with an as-built architecture, for instance in outsourcing situations as part of a product acceptance procedure

- Tracing of architecture elements to the source code, for instance to measure the impact of architectural changes
- However, only few organizations invest in SAR efforts. In the cases we are aware of, SAR is a facet in a much broader organizational context where software architectures play an important role in achieving particular business goals. Therefore it is worth understanding the organizational context and the role of software architecture to define an effective SAR concept.

From our extensive experience we have identified several contexts where SAR can be applied. Our experience includes: improving the understanding of the software architecture of existing systems, improving the architecture itself, assessing quality attribute characteristics of these systems, improving the documentation of the architecture of these systems, and more.

We experienced that the QADSAR approach is the right mechanism for driving reverse engineering of existing systems. Indeed, Tahvildari, et al., outline an approach that uses non-functional requirements or quality attributes, such as performance and modifiability to guide the reengineering process [37]. Bengtsson and Bosch outline a similar approach for reengineering based upon quality attribute scenarios that drive architecture transformation [4].

In our case we are applying a quality attribute driven approach to architecture reconstruction and goal-based system understanding. The goal of the reconstruction is to provide information that will assist in the analysis of the quality attributes.

In the past, several SAR efforts have related their work with more common/standard notations such as UML [31]. The goal of QADSAR is not to align architecture visualizations with mainstream notations, such as UML. The key to our approach is to enable architecture analysis

of existing systems via a quality driven approach. The analysis is motivated by the knowledge that software architectures are driven by business goals that are incorporating quality attribute scenarios.

The remainder of the paper is organized as follows. Section 2 outlines representative application contexts for SAR. Section 3 outlines the quality attribute driven (QAD) analysis framework that guides the SAR. Section 4 combines the QAD analysis with SAR and provides the fundamental QADSAR steps. In Section 5 we apply QADSAR in a real-world case study. Section 6 outlines our current research and future work. Section 7 concludes the paper.

2 APPLICATION CONTEXTS

From areas, such as migration to product lines, IT assessments, competing systems and system modernization, we have collected four representative contexts for applications of SAR:

- **Streamlining existing products into product lines.** Product lines embody a strategic reuse model of products sharing a market segment. One key practice in product lines is the definition of software product line architecture. Software architectures for product lines reflect common and variable parts of systems and offer appropriate design constructs. Product lines typically evolve out of the commonalities among existing products in a specific market segment. Several products may be delivered by an organization before a systematic migration to a product line takes place. In order to evaluate the potential for creating a product line from existing products it is necessary to analyze their architectures and compare the commonalities and differences across the product line candidates. Although SAR is not the primary intention of the organization, in this situation it offers a vehicle for architects to reconstruct the architecture in environments where it is poorly documented. This is done so as to be able to reason about software architecture commonalities and differences of existing products. Software architecture aspects of the current system are used as an input in the definition of the new software product line architecture. For elaborate treatments on product line migration refer to [12, 36].
- **IT assessments of existing systems.** Organizations evaluate the alignment of existing systems with business goals. Business goals have a strong influence on quality attributes, such as modifiability, performance, and availability. Often quality attributes are competing and result in tradeoffs. Changing business goals require a reassessment of tradeoffs, sensitivity points, and risks that are inherent in each system. Experiences show that an efficient assessment

approach is the analysis of quality attributes on a software architecture level [9]. It further allows the participation of various architecture stakeholders with specific roles, such as developers, project managers, and marketing experts. However, the architectures of existing systems may not be documented very well or may be inaccurate due to the erosion of design and implementation.

Again, SAR is not the primary concern that caused the initial effort at an organization. But SAR contributes to the overall goal by providing architecture documentation that is appropriate for the analysis of business goals.

- **Exclusive decision among competing existing systems.** A typical situation that is occurring quite frequently today is the merger of organizations with similar product portfolios. A decision for a favorable product is done depending on a set of criteria. Besides quantitative approaches, for example based on cost of producing or maintaining a product, there are qualitative approaches based on the software architecture comparison of the existing products [35]. The criteria for qualitative approaches are mainly quality attribute driven, for example the exchange of communication protocols (modifiability), or the processing of a particular number of transactions per day (performance). One of the difficulties in comparing existing software architectures is the heterogeneity of architecture descriptions in terms of terminology, concepts, architecture notations, and level of detail. To compare software architectures SAR provides a set of comparable architecture views in heterogeneous environments. These views are used for the quality-driven criteria analysis.
- **System Modernization.** Studies show that between 50% and 90% of software maintenance involves the understanding of the software being maintained [38]. One major reason for these high costs is due to architecture erosion which results in the maintainability of the software system being deteriorated. Software architects have to understand, analyze, and reason about the as-built software architecture of a system to modernize it [34]. SAR can support the understanding of existing systems. It allows software architects to form increasingly abstract models of a system and the resulting artifacts from SAR can be used to analyze quality-driven requirements that are caused by the demands of system modernization.

These application contexts don't represent a complete list.

Our purpose is to illustrate the common characteristics of application contexts in which SAR contributes to the overall goal. The characteristics are:

- Software architectures play a significant role within an organization. Software architectures are abstractions of implementations in the solution (design) space and at the same time the first implementations in the problem (requirements) space. At the transition of requirements and design they are recognized by various stakeholders of an organization as a major communication arena.
- The analysis of software architectures is quality attribute driven. Business goals are primarily incorporated as quality goals that shape the software architecture of a product. Evaluating quality attributes reveals the tradeoffs and risks that are involved in architecture design decisions.
- Software architecture documentation, if it exists at all, often describes the as-built architectures insufficiently and in many cases inaccurately. A precondition for reasoning about software architectures and quality attributes is the existence of good architecture descriptions that provide accurate information about the as-built systems.

All three characteristics have to be considered upfront to an application of SAR. Figure 1 illustrates the informal connection between SAR and the application contexts, assuming that a SAR is adding value. The effort is divided into two parts: reasoning and reconstruction. The reasoning part consists of the application context and the QAD Analysis Framework. The analysis framework is a means to evaluate systems in the achievement of particular quality attribute goals, for example security and scalability goals. The analysis framework steers the architecture reconstruction to make particular aspects of existing software visible. For example, a performance model might require threads, queues, waiting points, and performance properties such as throughput and deadlines. SAR has to provide the information inquired by the QAD Analysis Framework.

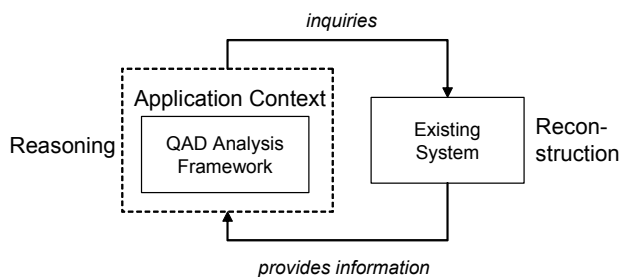


Figure 1: Reasoning and Reconstruction

Figure 1 also illustrates the goal of an analysis driven approach that is founded in the informal connection

between reasoning and reconstruction:

The goal is to assist software architects in the analysis of quality attribute driven goals in poorly documented architecture settings.

The goal has implications on criteria for completeness, the types of elements to be reconstructed, and criteria for poor documentation that result in a SAR effort:

- **Completeness.** Reconstructing software architectures require criteria for completeness: How many architecture views have to be reconstructed to sufficiently describe the architecture? The yardstick for completeness is the analysis framework. The goal is not to reconstruct every architecture aspect of a software system but to provide the necessary information required by targeted quality attribute models of the analysis framework.
- **Types of elements.** A question arising in many SAR efforts relates to the type of elements, relations, and notations that should be reconstructed: Is it sufficient to describe the architecture in layers represented as box-and-arrow drawings? Again, the approach answers this question from the perspective of the analysis framework. For example, performance formulas require specific performance data and may not take into account layers and graph aspects at all. The element types, relations and particular properties, such as throughput, are given by the quality attribute models of the analysis framework.
- **Re-documentation.** The criteria for poor documentation are given by the ability of the documentation to provide sufficient information to the analysis framework: Are required element types sufficiently described to feed a particular quality attribute model? Software architecture documentation might sufficiently describe element types of a set of quality attribute models (for example, analyzed in the design process) but may lack element types of a model that is currently under investigation.

3 QAD ANALYSIS FRAMEWORK

The application contexts introduced above require the analysis of existing systems. The analysis is driven by business goals, articulated in quality attributes that should be evaluated on existing systems. Evaluations require a systematic way to reason about the achievement of quality goals. We denote the “systematic way” as a framework that guides software architects to evaluate or design architectures. The analysis framework that we refer to in our approach adopts the architecture analysis concepts as proposed in [2].

Quality attributes are refined into quality attribute scenarios.

“A quality attribute scenario is a quality attribute

requirement of a system. It consists primarily of a stimulus and a response. The stimulus is a condition that needs to be considered when it arrives at a system and the response is the (measurable) activity undertaken after the arrival of the stimulus. In addition to the stimulus and response, a quality attribute scenario includes the source of the stimulus, the context under which the stimulus occurs, the artifact that is stimulated, and how the response is to be measured” [3].

An example scenario is: *The system has to be able to accept and execute user commands 250 ms after power-on.* The source of the stimulus is the *power-on switch*, the stimulus is *power-on*, the context is *startup*, the response is *normal mode of operation*, and the measure is *250ms until user input can be accepted*.

Quality attribute scenarios are input to a corresponding Quality Attribute Model, such as a performance model. To achieve particular qualities that are addressed with scenarios, developers decide to structure the software in a particular way. For example, to meet the 250ms deadline in the above scenario, a designer might decide to use an architecture tactic such as *reduce computational overhead* by avoiding in-depth memory checks during start-up.

The authors of [2] introduce the notion of tactics as a means to control a quality attribute response by manipulating some aspect of a quality attribute model through architecture design decisions. A tactic is an architecture strategy that “is concerned with the relationship between design decisions and a quality attribute response”. There are collections of tactics available to achieve particular quality attribute goals (for further details refer to [3]). Different quality attributes have quality attribute models with different precision. For example performance models can be fairly formal (queuing models) while usability models try to model user satisfaction, which can be very informal.

Quality Attribute Models can be used in the design as well as in the analysis of software systems. The context in case of an existing system is distinguished from a design process in the following ways:

- The architecture tactics are not free to select. In the design process the software architect has to select the appropriate tactic to satisfy the required responses. In the case of an existing system the tactics are already realized and have to be extracted from existing sources. The extraction is guided by proposed lists of tactics for each quality attribute model. These lists can be obtained from [3].
- Each Quality Attribute Model requires particular architecture elements and their properties. For instance, architecture elements for a performance model are units of concurrency which can be implemented as processes and threads. Properties

represent specific information regarding an element, such as throughput, thresholds, deadlines, and race conditions. Again, elements and properties in the analysis process are obtained from the existing system. Note, that the terminology of elements and properties of Quality Attribute Models might differ from the terminology already used in the existing system.

- The responses of the Quality Attribute Models for elements and properties of an existing architecture are used to determine if the existing system or part-system satisfies the requirements as provided by the quality scenarios.

Figure 2 illustrates this connection. The QAD Analysis Framework steers the inquiries from the existing system to feed the Quality Attribute Model with the required architecture elements.

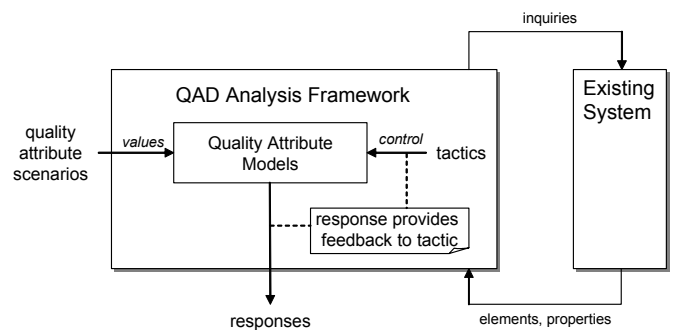


Figure 2: Analysis Framework

At this point we would like to subsume architecture elements, relations, properties, and tactics under the concept of architecture views. An architecture view is a *representation of a set of system elements and relations among them* [8]. An architecture view has a defined notation that specifies element types (such as component, task, or pattern), relation types (such as ‘consist of’, or ‘is-a’), and property items (such as throughput values).

4 COMBINING QAD WITH SAR

So far we considered SAR as a black-box that has to provide information about existing systems to support the QAD Analysis Framework. The application contexts of Section 2 presented some stumbling blocks:

- Architects of existing systems may no longer be available,
- the system is eroded from the as-designed architecture, and
- the documentation contains poor information related to a quality attribute model under investigation.

To tackle the architecture view extraction the following four strategies (ST) might apply depending on the application context:

- ST1 - Extract the required information from available architecture documentation. The success of this activity depends on the documentation quality and the relevance of the contained information in regards to the as-built system.
- ST2 - Interview the expert. This is probably the easiest way to obtain answers if the expert is still available.
- ST3 - Conduct a reconstruction workshop. Architectures have several stakeholders with expertise in particular areas. The workshop typically sets at the beginning a common understanding of software architectures and the purpose of reconstruction before the information is extracted in a group effort.
- ST4 - Undertake an architecture reconstruction from available sources, such as source code. This strategy is the most labor-intensive strategy. It requires a mixture of all strategies above in addition to the source elicitation. However, the results are more accurate, reflect the as-built architecture, and offer traceability from the code back to the design.

Our approach follows the last strategy on the basis of an as-built system. However, it is a guided extraction process in the way that not all information is elicited but only the elements and properties relevant for the analysis framework.

Applications of SARs are performed in various ways because SAR is embedded in a broader organizational context. For example, SAR in a product line migration context might be performed over promising assets of existing systems [12]. SAR in an architecture conformance setting [35] reconstructs the views that are specified in an as-designed architecture document [24]. In any case, there is a set of core steps that is used in most SAR applications. The steps comprise:

- Step 1: **Scope Identification.**
- Step 2: **Source Model Extraction.**
- Step 3: **Source Model Abstraction.**
- Step 4: **Element and Property Instantiation.**
- Step 5: **Quality Attribute Evaluation.**

Figure 3 provides an overview of the steps. The trigger for the method is the QAD Analysis Framework that requires architecture information to perform the quality attribute analysis. Step 1 (*Scope Identification*) sets the scope for SAR. The scope identifies the architecture viewtypes [8] and the part (or parts) of the system (or systems) that should be reconstructed. The identification depends on the quality attribute scenarios, the related quality attribute models, and the type of system.

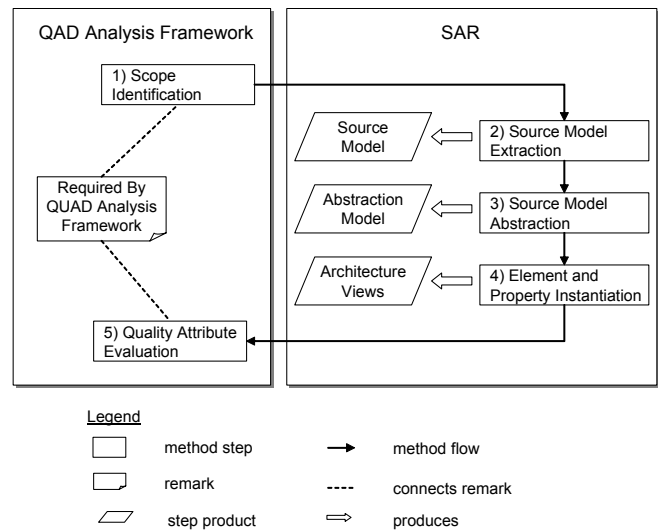


Figure 3: The QADSAR Steps

The approach continues with step 2 (*Source Model Extraction*) to extract source elements from available sources. Source elements are typically the constructs of the implementation language like functions, classes, files, and directories. Relations describe how the source elements relate to each other, such as call relations between functions or read accesses by methods on attributes. Besides static aspects there are also dynamic aspects like function execution time, or process relations. The static relations are typically generated by existing tools like source code parsers or lexical analyzers. Dynamic information is generated by profiling or code instrumentation techniques. The extracted elements and relations constitute the *Source Model*.

Elements of the source model are in most cases too fine-grained for architecture reasoning. Therefore, step 3 (*Source Model Abstraction*) has to identify and apply aggregation strategies to abstract from detailed source views. There exist several aggregation strategies [Harris 95 WCRC], which highly depend on the existing system and the architecture views that should be extracted. Various techniques exist such as Relation Partition Algebra and Tarski Algebra for manipulation of relational information [13,14,19]. A common abstraction technique is aggregation of coherent functionality. Other techniques capture independent branches in the calling graph or aggregate functions attached to an execution process. There could be low-level aggregation techniques like collection of all files in a directory or extracting files and functions following certain naming conventions. The aggregated elements constitute the *Aggregation Model*.

The aggregation model consists of entities and relations that are collapsed. They might be associated with architecture elements but they are not explicitly denoted as architecture elements with particular properties. To obtain the required architecture views we have to assign in step 4

(Element and Property Instantiation) the element types specified by the view-type of the analysis framework. Elements are now layers, tasks, ‘consist of’ relations, etc. We have to assign required properties, such as throughput, deadlines for tasks, etc. Further on we would like to associate tactics that are achieved with a particular set of architecture elements.

The results of step 4 feed the QAD Analysis Framework for step 5 (*Quality Attribute Evaluation*) that is performed with the particular quality attribute scenarios, quality attribute models, and the corresponding architecture tactics. The tactics are used to detect mechanisms in the reconstructed views that support the quality attribute scenarios.

5 CASE STUDY

To illustrate the usage of QADSAR we present a case study that we performed in the domain of automotive body components, such as window lifters, climate controls, sunroofs, and door locks. The broader context of the case study is embedded in a multi-year effort to streamline automotive body products into product lines. Product lines require from organizations strategic usage of practice areas in software engineering, technical management, and organizational management [7]. A key practice area is ‘software architecture definition’ which produces the first design artifacts that begin to place requirements into a solution space. Besides a top-down approach to design software architectures based on domain analysis efforts and sets of specific product requirements and business goals the architects wanted to explore and analyze the software architecture of a representative set of existing products.

The analysis was done on an automotive body door unit that was at the beginning of the case study project already at a testing stage. A Door Control Unit (DCU) is the hardware/software unit that is physically located in both the front and rear driver- and passenger-side automobile doors. The DCU provides a rich set of features, such as the control of window positions, exterior mirror position, interior lighting, seat-belt indication, door-open indication, exterior door lamps, low-battery indication, seat positions and seat temperature control. The units for each door are networked and arbitration is performed between (possibly conflicting) requests from different units.

We apply in the following the reconstruction steps as proposed in Section 4.

Step 1 - Scope Identification

The DCU consists of three packages from different vendors: Boot Loader, Communication, and Application. The communication package is typically predetermined by the OEM (Original Equipment Manufacturer) and has to be deployed by all suppliers that provide networked devices for a specific model platform. Suppliers, following a product line approach, are interested in keeping their

application software adaptable to the communication software predetermined by the OEM. Therefore, one task of this project is to investigate the modifiability of the application software with regards to other communication software packages.

The concrete quality attribute scenario for the analysis is: The organization has to replace the communication package from vendor V1 by a package from vendor V2 in one day. The source of the stimulus is the *organization*, the stimulus is *organization requires to exchange communication package*, the context is during *deployment time*, the response is *modification without affecting other functionality*, and the response measure is *number of modules affected and effort*. To determine the viewtypes that we require for the analysis we still need to construct the quality attribute model for modifiability.

Modifiability is strongly influenced by the different types of dependencies between modules of a system. To analyze the impact of a change the knowledge of these dependencies is of importance. *A dependency among modules exists, if a modification to some aspects of module A requires a modification in module B to accommodate the modification to module A. We then say that module A depends in some way on module B.* Here is a list of possible dependency types that we summarize from [3]:

- **Syntax dependencies**

Syntax dependencies between modules A and B can be either data (type/format of data is consistent) or service (signature of services are consistent) related.

- **Semantics dependencies**

Semantic dependencies between module A and B can be either data or service related.

- **Sequence-of-use dependencies**

Sequence-of-use dependencies can be either data or control related.

- **Interface identity dependencies.**

Interfaces between module A and B must be consistent (same name or handle).

- **Runtime location dependencies.**

Runtime location dependencies (on same or different processor or located within different processes) between module A and B must be consistent.

- **Quality-of-service or quality of data dependencies.**

Quality-of-service or quality of data dependencies involve the service or data provided by the modules.

- **Existence-of-module dependencies.**

Existence-of-module dependencies involve module A

or B being present for the other to function properly.

- **Resource behavior dependencies.**

Resource behavior dependencies relate to resource behavior (such as memory usage, resource ownership) between module A and module B.

A quality attribute model for modifiability is more informal than a model for performance that could rely, for example, on rate monotonic scheduling or queuing models [1].

From the quality scenario, the dependency types, and the type of system under investigation we are able to determine the viewtypes. We select from the more general dependency list above the dependency types that are useful for the DCU software and the particular quality attribute scenario.

- Syntax dependencies - both, data and functions with parameters can be elicited from a module view.
- Semantic dependencies – Contracts about content (data and service) are difficult to extract. However, a good point to start is the analysis of denoted interfaces with semantic descriptions.
- Sequence-of-use-dependencies – for data: dataflow views; for service: interaction diagrams or state machine views.
- Interface identity dependencies – not of relevance in this context.
- Runtime location dependencies – typically presented in a deployment view. Location dependencies exist in terms of different configurations for driver and passenger side. However, for the particular communication package it is of no further interest.
- Quality-of-service –or quality of data dependencies, Existence of module dependencies, and resource behavior dependencies, that require concurrency views, are not further considered in this paper. However, it is obvious, that a modifiability analysis is frequently not independent from a performance analysis. For example, resource behavior might require scheduling assumptions. An exchange of a communication package following a rate monotonic scheduling might require different execution time budgets.

Due to space and illustration purposes we concentrate on the analysis by using several module views.

Modifiability might be achieved in a software system by different architecture tactics. Examples to achieve modifiability for the quality attribute scenario of this case study are: maintain semantic coherence, isolate expected changes, and hide information. Examples for other modifiability tactics are: use a virtual machine, limit

communication paths, or abstract common services. Common to the tactics are strategies to reduce and manage the dependencies (as previously enlisted) between modules.

The tactics help us in the following reconstruction process to identify what to look for in the existing system and to create particular views. Applied to this context:

- Maintain semantic coherence – this tactic has not to be evaluated because the communication package is already determined to be separated by the OEM.
- Isolate expected changes – that is: is something actively done in the application package to mitigate changes.
- Hide information – for example, is there an explicit model to separate interfaces from their realizations.

Step 2 - Source Model Extraction

Source model extraction is a well-known step in architecture reconstruction ([10,31]). There are many program tools available to extract the information needed for a source model ([20,33,42]). The tools used depend on the language in which the system is implemented, which in this case is C. From the type of the language in which the system is implemented, and the required viewtypes, we can identify a set of elements (such as files, variables, functions etc.) and relations (such as calls, contains, includes, etc.) to extract.

Most of the extraction tools provide the ability to output their analysis results in a text file, which can be manipulated using a scripting language, such as perl [40,41], into the format required by a reconstruction tool. One format frequently used is the Rigi Standard format (RSF) [28] (a tuple-based data format in the form of “relation <entity1> <entity2>”).

The extracted information was loaded into our experimental reconstruction tool ARMIN (**A**rchitecture **R**econstruction and **M**INing), that we are currently building for QADSAR. The Figures (5, 6, 7, and 8) are screenshots presented by ARMIN. The figure notation, such as relation conventions, is analogous to Rigi [30].

Step 3 - Source Model Abstraction

The reconstruction of any system involves several activities for abstracting from the source level information, which consists of the set of elements and relations, to higher-level views of that information. These activities include, for example, aggregation, pattern matching, analyzing documents to assist in identifying abstractions, and interviewing developers and maintainers to assist in identifying abstractions. Figure 4 illustrates the different abstraction levels. The “File+” aggregation subsumes local information inside files (such as static functions and local variables) to hide details that are not architecturally relevant. Files are composed to sub-layers (L-1), layers (L-0), and finally the DCU system (System) according to the

as-designed software specification.

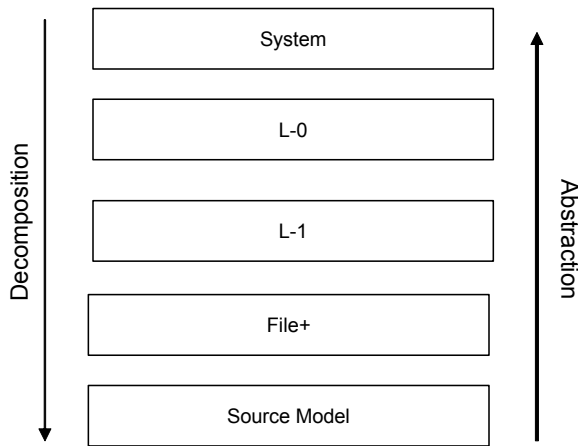


Figure 4: Abstraction Model

The resulting aggregation is illustrated in Figure 4. The content of the collapsed relations between the application and the communication package are illustrated in Figure 5.

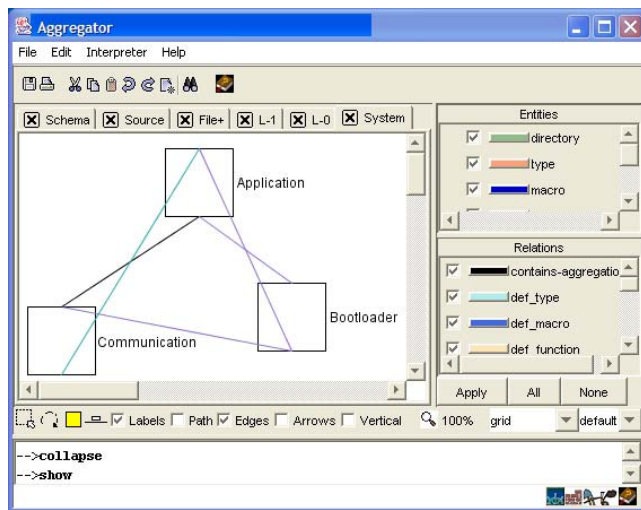


Figure 5: Software Packages

Parts of the content of the collapsed edge between the Application and the Communication package are illustrated in Figure 6. The top directory (*Application -> Communication*) is part of the system abstraction level of Figure 4. Below that is the L-0 level (for example *Data->LAN*) followed by the L-1 and File+ levels. At the lowest level are, for example, calls relations (*FrontDoorReq -> WindowMoveReq*). Figure 6 allows a detailed syntax dependency analysis.

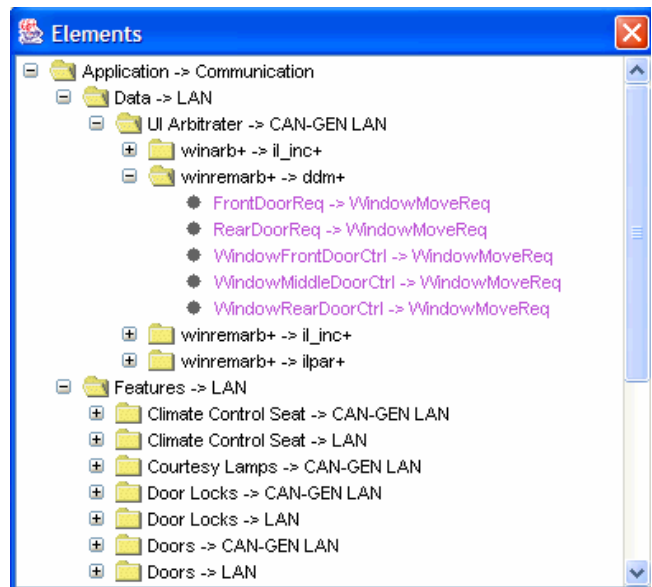


Figure 6: Module View - Syntax Dependencies

To detect the 'hide information' tactic we try to isolate from the application specific interface files that manage the access to and from the communication package. The result is illustrated in Figure 7.

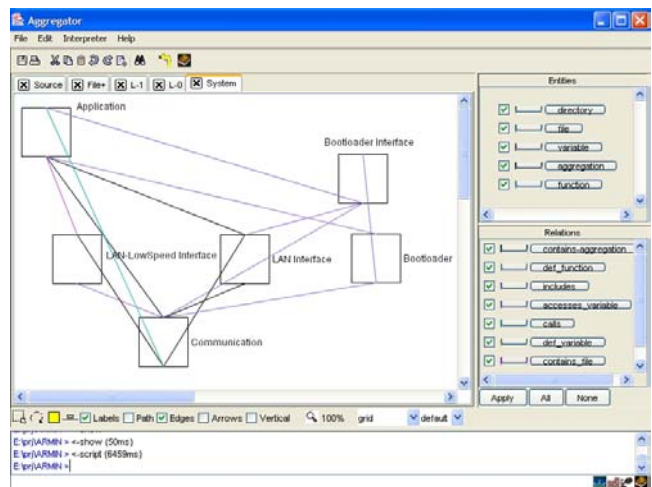


Figure 7: Module View - Interface Identification

Finally, the quality attribute scenario identified in step 1 expects a response measure in terms of 'number of modules affected'. For this we extract a dependency layout for the LAN part of the communication package, as illustrated in Figure 8 and analyze the application modules that access the 'LAN' module. All modules above 'LAN' are accessing the 'LAN' component unidirectional. The 'Diagnosis' and 'uC_Config' modules have bidirectional relations whereas the LAN Common part accesses 'LAN Common'. Note, that Figure 8 doesn't show relations among modules other than 'LAN'.

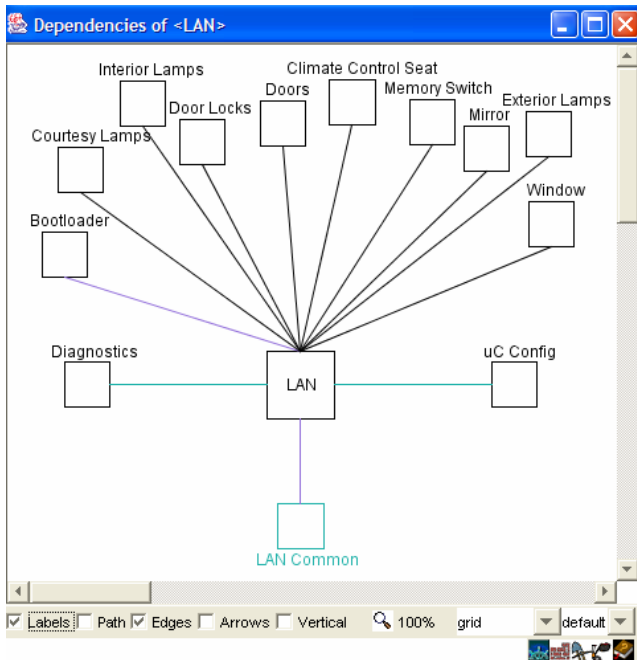


Figure 8: Module View - Affected Modules

Step 4 – Element and Property Instantiation

The generated entities and relations of step 3 might be associated with architecture elements, such as layers, subsystems, design patterns, etc., but they are not explicitly denoted as architecture elements with particular properties. This capability is important, for example, to export models to formal performance analysis tools, such as TimeWiz [39]. For the modifiability part of this case study it is sufficient to use the generated entities and relations from step 3.

Step 5 – Quality Attribute Evaluation

Using the module views from step 4, the quality attribute model for modifiability and the quality attribute scenario from step 1 with the set of modifiability tactics, we are now able to perform the evaluation. Strictly speaking, the evaluation already started in step 4 with the generation of the module views.

The two tactics that could support the communication package exchange are: isolate expected changes and hide information. Both tactics reduce the dependencies and the number of modules affected by changes. However, both tactics are not identifiable in the analysis of Figures 7 and 8.

Figure 7 shows that there is no explicit notion of interface identifiable, e.g. no separation of interface and implementation. Several attempts to capture an interface by different aggregations failed. An analysis of Figure 8 with the developers illuminated that changes are likely to be expected in more than a dozen modules.

The case study started with the analysis of quality attribute

scenario that captured one of the business goals of the organization. Based on the scenario and type of system we developed a quality attribute model and defined the required architecture viewtypes and elements. The specific tactics for modifiability were used to detect mechanisms in the existing system that support the business goal. The final evaluation resulted in an improvement effort of the existing system to support the quality attribute scenario sufficiently.

6 CURRENT AND FUTURE WORK

Based on our work in architecture reconstruction, software product lines and software architecture analysis, we are currently undertaking research to address the demands of formal and informal quality attribute analysis of existing systems by incorporating software architecture reconstruction techniques. We are currently doing and plan to continue work on:

- Investigating the relationships between the quality attribute analysis and the architecture views that would need to be reconstructed to support the analysis.
- Developing a support tool (ARMIN) for QADSAR with export mechanisms to use specialized analysis tools for particular quality attributes, such as performance tools [39]. ARMIN is a successor for the Dali Architecture Reconstruction Workbench [10].

7 CONCLUSIONS

The QADSAR approach establishes the link between Quality Attribute Driven Analysis and architecture reconstruction. The business goal driven approach of system understanding provides an efficient way to steer the reconstruction process by providing the required viewtypes for a particular system. The quality attribute related architecture tactics are an efficient way to infuse the reconstruction and analysis process to measure the response for particular quality attribute scenarios. The response measure is linked back to the business goals of an organization.

The case study has shown the application of QADSAR for a modifiability scenario in an embedded system. Applications for further quality attribute scenarios are derivable with the techniques of Quality Attribute Models and their related architecture tactics.

We believe that from a detailed investigation of a particular quality attribute and from the type of system involved QADSAR provides a substantial contribution to leverage SAR in a concrete organizational context.

REFERENCES

1. Bachmann, F.; Bass, L. and Klein, M. Illuminating the Fundamental Contributors to Software Architecture Quality, *CMU/SEI-2002-TR-025*, Software Engineering Institute, Carnegie Mellon University, 2002

2. Bachmann, F.; Bass, L. and Klein, M. Deriving Architectural Tactics – A Step toward Methodical Architectural Design, *CMU/SEI-2003-TR-004*, Software Engineering Institute, Carnegie Mellon University, 2003
3. Bass, L.; Clements, P. and Kazman, R. *Software Architecture in Practice*, 2nd Edition. Reading MA: Addison Wesley, 2003.
4. Bengtsson, P. and Bosch, J., Scenario-based Software Architecture Reengineering, *Proceedings of the 5th International Conference on Software Reuse (ICSR5)*, IEEE, pp. 308-317, 2-5 june, 1998.
5. Bowman, T.; Holt, R. C. and N. V. Brewster, Linux as a Case Study: Its Extracted Software Architecture. *Proceedings of the International Conference on Software Engineering*, Los Angeles, May 1999.
6. Buschmann, F.; Meunier, R.; Rohmert, H.; Sommerlad, P. and Stal, M., *Pattern-Oriented Software Architecture*, New York, NY: John Wiley & Sons, 1996.
7. Clements, P. and Northrop, L., *Software Product Lines*, Addison Wesley, 2002
8. Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R. and Stafford, J., *Documenting Software Architectures: Views and Beyond*, Addison Wesley, 2002.
9. Clements, P.; Kazman, R. and Klein, M., *Evaluating Software Architectures*, Addison Wesley, 2002.
10. The Dali Architecture Reconstruction Workbench: http://www.sei.cmu.edu/ata/products_services/dali.htm
11. Eixelsberger, W.; Ogris, M.; Gall, H. and Bellay, B., Software architecture recovery of a program family, *Proceedings of the International Conference on Software Engineering*, pp 508 –511, Kyoto Japan, April 1998.
12. Faust, D. and Verhoef, C. Software Product Line Migration and Deployment, *Software – Practice & Experience*, Published by John Wiley & Sons, Ltd, 2003.
13. Feijs, L. M. G. and Krikhaar, R. L., Relation Algebra with Multi-Relations, *International Journal of Computer Mathematics*, 70, pp 57-74, 1999.
14. Feijs, L. M. G. and van Ommering, R. C., *Theory of Relations and its Applications to Software Structuring*, *Phillips Research Internal Report*, 1994.
15. Finnigan, P. J.; Holt, R.; Kalas, I.; Kerr, S.; Kontogiannis, K.; Mueller, H.; Mylopoulos, J.; Perelgut, S.; Stanley, M. and Wong, K., *The Portable Bookshelf*, *IBM Systems Journal*, Vol. 36, No. 4, pp. 564-593, November 1997.
16. Guo, G.; Atlee, J. and Kazman, R., A Software Architecture Reconstruction Method, *Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1)*, San Antonio, Texas, February 22-24, 1999 pp 225-243.
17. Harris, D.R.; Reubenstein, H. B. and Yeh, A. S., Recognizers for Extracting Architectural Features from Source Code, *Proceedings of the 2nd Working Conference on Reverse Engineering*, 1995.
18. Harris, R.; Reubenstein, H. B. and Yeh, A. S., Reverse Engineering to the Architectural Level, *Proceedings of the International Conference on Software Engineering (ICSE)*, pp 186-195, April 1995.
19. Holt, R., Structural Manipulations of Software Architecture using Tarski Relational Algebra, *Proceedings of the Working Conference on Reverse Engineering*, Honolulu, Hawaii, pp 210-219, October 12-14 1998.
20. Imagix Corporation's Imagix 4D: <http://www.imagix.com/>
21. Kazman, R. and Carrière, S. J., Playing Detective: Reconstructing Software Architecture from Available Evidence, *Journal of Automated Software Engineering*, pp 107-138, April 1999.
22. KLOCwork inSight: <http://www.klocwork.com/Accelerator.htm>
23. Krikhaar, R. L., *Software Architecture Reconstruction*, *Ph.D. Thesis*, University of Amsterdam, 1999.
24. Kruchten, P. The '4+1' View Model of Software Architecture, *IEEE Software* 12, 6, November 1995.
25. Laine, P. K., The Role of Software Architecture in Solving Fundamental Problems in Object-Oriented Development of Large Embedded Systems, *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*. Amsterdam, The Netherlands, pp 14-23, August 28-31, 2001.
26. Lassing, N., *Architecture-Level Modifiability Analysis*, *PhD Thesis*, Vrije Universiteit Amsterdam, 2002, <http://www.cs.vu.nl/~nlassing/research/thesis.pdf>
27. Mendonça, N. C. and Kramer, J., Architecture Recovery for Distributed Systems, *SWARM Forum at the Eight Working Conference on Reverse Engineering*, Stuttgart, Germany, October 2001.
28. Müller, H. A.; Mehmet, O. A.; Tilley, S. R.; and Uhl, J. S. *A Reverse Engineering Approach to System Identification*. *Journal of Software Maintenance*:

Research and Practice 5, 4, December 1993.

29. The Portable Bookshelf: <http://swag.uwaterloo.ca/pbs/>
30. The Rigi Tool: <http://www.rigi.csc.uvic.ca/>
31. Riva, C., Reverse Architecting: An Industrial Experience Report, *Proceedings of the Seventh Working Conference on Reverse Engineering*, Brisbane, Australia, pp 42-50, November 23-25, 2000.
32. Sartipi, K. and Kontogiannis, K., A Graph Pattern Matching Approach to Software Architecture Recovery, *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2001)*, Florence, Italy, pp 408-419, November 7-9, 2001.
33. Scientific Toolworks Inc's Understand for C/C++/Java /Fortran/Ada: <http://www.scitools.com/>
34. Seacord, R. C.; Plakosh, D.; and Lewis G. A., *Modernizing Legacy Systems*, SEI Series in Software Engineering, Addison Wesley, 2003.
35. Stoermer, C.; Bachmann, F. and Verhoef, C, SACAM – The Software Architecture Comparison Analysis Method, *CMU/SEI-2003-TR006*, (to be published).
36. Stoermer, C. and O'Brien, L., MAP- Mining Assets for Product Line Evaluations, *Proceedings of the Third Working IFIP Conference on Software Architecture (WICSA 01)*. Amsterdam, Netherlands, 2001.
37. Tahvildari, L.; Kontogiannis, K. and Mylopoulos, J., Quality-driven software re-engineering, *Journal of Systems and Software*, vol. 6. Issue 3, June 2003.
38. Tilley, S. and Smith, D. B. Perspectives on Legacy System Reengineering, Software Engineering Institute, Carnegie Mellon University, 1995, available at: <http://www.sei.cmu.edu/reengineering/lsysree.pdf>
39. TimeWiz: <http://www.timesys.com>.
40. Wall, L. and Schwartz, R. L., *Programming Perl*, O'Reilly & Associates, Inc., 1991.
41. Wall, L.; Christiansen, T. and Schwartz, R. L., *Programming Perl*, 2nd Edition, O'Reilly & Associates, Inc., 1996.
42. Windriver's SNiFF+: <http://www.windriver.com/>