QUALITY ANALYSIS OF A CROSS-DOMAIN REFERENCE ARCHITECTURE

Liliana Dobrica
Faculty of Automatic Control and Computers, University Politehnica of Bucharest
Spl. Independentei 313, Romania

Eila Ovaska
VTT Technical Research Centre of Finland, Oulu, Finland

Keywords: Cross Domain Reference Architecture, Service, Quality, Analysis Methods, Scenarios.

Abstract: The content of this paper addresses the issue of how to perform in a systematic way quality analysis of a cross domain reference architecture using scenarios. The cross domain reference architecture is designed based on the domains requirements and features modelling and it includes domains core services and constraints on how these services should be combined. We apply a method based on scenarios to analyse modifiability at the architectural level. In order to handle complexity in analysis we propose categories of change scenarios to be derived from each problem domain. Our main concerns are core services changes in the scenarios interaction step.

1 INTRODUCTION

Nowadays many systems are used as subsystems in various application domains. Due to the escalating complexity level and the higher competition in the world market, a coherent and integrated development strategy is required. It becomes a research priority the creation of a generic architecture and a suite of abstract components with which new developments in different application domains can be engineered with minimal effort. Generic architecture can be based on a core architectural style. Given a core architectural style, different components are created for different application domains, while retaining the capability of component reuse across these domains. The goal of architectural analysis is to get measures of compliance with regard to requirements specification (Dobrica and Niemela, 2002). It is very important to identify which are the relevant properties of each domain and how analysis techniques and methods could be applied to a cross domain reference architecture (RA). There are two categories of properties related to software systems, the general one, like performance, satisfaction of real-time requirements, reliability, etc. and specific to development process (Bosch, 2000). Among the specific properties that deserve special attention are kinds of variation which can be covered by the architecture and properties that are preserved for all variants of an architecture in specific domains, stability of services interfaces with respect to evolution in products.

The open problem of an architectural analysis method is how to take better advantage of architectural concepts to analyze for quality attributes in a systematic way (Bass et al., 1998). The RA must be generic and adaptable to the multiple composed domains. One objective of the evaluation is to minimize possible changes in functionality required by various domain specific services. It is also very important to identify potential risks and to verify that the quality requirements of the embedded systems domains have been addressed in the RA design (Graff et al., 2005). Analysis could be associated with the design in an iterative improvement of the RA when the system of systems is initiated from requirements specification, or for the re-engineering of an existent complex embedded system due to the evolution process.

Modifiability is one of the main quality drivers for cross-domain RA design. The analysis of this quality attribute could be combined with other run-time quality requirements (performance, reliability, security, etc.) of the domains. In this paper we evaluate modifiability of a cross-domain reference architecture for embedded systems applications. In the next section we define the cross domain approach for architecture development, then we discuss about an appropriate quality analysis method. Finally we explain our view on a concrete example and we perform modifiability analysis. Our aim is to associate design with analysis in an iterative improvement of the complex systems architecture. We argue with experiences in the software architectures design and analysis for various domains (Dobrica and Niemela, 2008) (Dobrica and Ovaska, 2009) and other researchers' recent studies that will be revealed during the paper.

2 BACKGROUND

A system is a collection of cooperating services that deliver required functionality. These services may be executed in a networked environment and may be recomposed dynamically. A cross domain approach consists of three levels for architecture development of a software system (Dobrica and Ovaska, 2009) (Figure 1). The RA level includes core services and focuses on commonality analysis. Also the RA includes rules or constraints on how core services should be combined to realize a functional goal. Domain architecture level includes domain specific services and requires variability management concerns. The last level is for product architectures. On this level rules for product derivation and configuration are included.

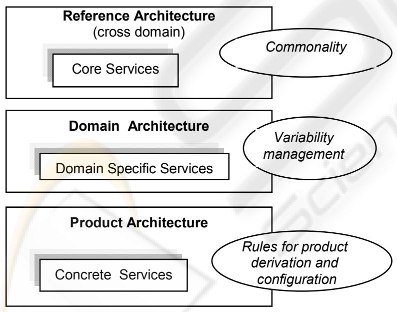

Figure 1: Architecture development approach.

A feature model is a prerequisite of design (Dobrica and Ovaska, 2009). This model is essential for both variability management and product derivation, because it describes the requirements in terms of commonality and variability, as well as defining dependencies. Features may be mandatory, optional, alternative or optional alternative. The features model specifies dependencies called composition rules. The *requires rule* expresses the presence implication of two features and the *mutually exclusive rule* captures the mutual exclusion constraint on feature combinations.

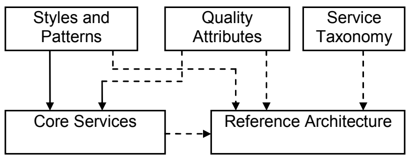

Figure 2: Reference architecture realization.

RA is defined by quality attributes, architectural styles and patterns and abstract architectural models (Figure 2). *Quality attributes* clarify their meaning and importance for core service components. Services have to meet many quality attributes. Interdependencies and tradeoffs also exist between them. The *styles and patterns* are the starting point for architecture development. Architectural styles and patterns are utilized to achieve qualities. A style defines a class of architectures and is an abstraction for a set of architectures that meet it. A pattern is architectural when it is a documented description of a style or a set of styles that expresses a fundamental structural organization schema applied to high-level system subdivision, distribution, interaction, and adaptation (Buschman et al., 1996). In this way the RA creates the framework from which the architecture of new products is developed. It provides generic architectural services and imposes an architectural style for constraining specific domain services in such a way that the final product is understandable, maintainable, extensible, and can be built cost-effectively. Potential reusability is highest on RA level. Core services and the architectural style are reused in every domain. RA is build based on a *service taxonomy,* that defines the main categories called domains. Root features that have been abstracted from requirements characterize services. A service taxonomy guides the developers on a certain domain and getting assistance in identifying the required supporting services and features of services.

3 ANALYSIS METHOD

Scenario-based assessment is particularly appropriate for qualities related to software development. Software qualities such as maintainability, reusability, modifiability, adaptability and portability can be expressed very naturally through change scenarios. The use of scenarios for evaluating architectures is recommended as one of the best industrial practices

(Kazman, 2000). Our method is based on the SAAM (Kazman et al., 1996), but improved through the introduction of guidelines for analysis. The method consists of five important steps. These are:

1. Deriving of Change Categories from the Problem Domain. Figure 3 presents five categories of the change scenarios derived from the problem domains. It may be possible that a change scenario related to one of these categories requires other changes in the other categories. It is recommended to consider this possibility in the scenario development process. Usually it occurs when the problem domain is organized so that it is easy to identify the main sources for the addition of subsequent features in the domain.

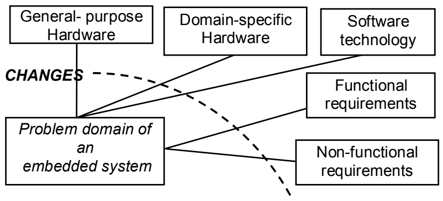
General- purpose Hardware
Domain-specific Hardware
Software technology
CHANGES
Functional requirements
Problem domain of an embedded system
Non-functional requirements

Figure 3: Categories of scenarios.

2. Scenarios Identification. Scenarios should illustrate the kinds of anticipated changes that will be made to the system. A common problem of the scenario development is when to stop generating scenarios. Using a set of *standard quality attribute-specific questions* we ensure proper coverage of an attribute by the scenarios. The boundary conditions should be covered. A standard set of quality-specific questions allows the extracting of the information needed to analyze that quality in a predictable, repeatable fashion. For analyzing the modifiability we must look for possible changes in the problem domain defined requirements.

3. Architecture Description could be performed in parallel with the previous one. Architecture description may use multiple views. For a common level of understanding a small and simple lexicon could be used in describing structures.

4. Evaluate the Effect of the Scenarios on the Architecture Elements. The effect is estimated by investigating which services are affected by that scenario. The cost of the modifications associated with each change scenario is predicted by listing the services that are affected and counting the number of changes. The objective is to get a measurement of the quality of the core and domain services with respect to the anticipated variability in functional or non-functional characteristics.

5. Scenario Interaction. The result of the effects evaluation represents the input for this step. The activity is to determine which scenarios interact, meaning that they affect the same service. High interactions of unrelated scenarios indicate a poor separation of concerns. Also if any of the scenarios affect a core service this is no more part of the RA, but a domain specific.

4 THE CASE STUDY

An analysis method is very difficult to discuss on an abstract level. Instead, one needs a concrete example. In this section we present a case study of cross domain reference architecture. Our example is abstracted from our experiences with the architecture design of a scientific on-board silicon X-ray array (SIXA) spectrometer control software.

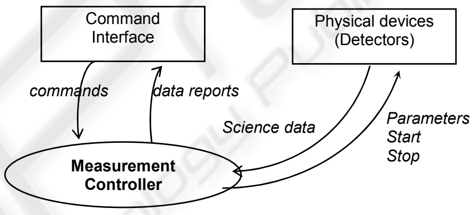

Figure 4: Context view of the required system.

Figure 4 introduces the context view of a measurement controller. External elements that interface with our measurement controller are a command interface and physical devices (detectors) representing sensors and actuators. The system is programmed and operates using a set of commands sent from a command interface. The role of the spectrometer controller is to control the following measurement modes: (a) Energy Spectrum (EGY), which consists of three energy-spectrum observing modes: Energy-Spectrum Mode (ESM), Window Counting Mode (WCM) and Time-Interval Mode (TIM). (b) SEC, which consists of single event characterization observing modes: SEC1, SEC2 and SEC3. Each mode could be controlled individually. A coordinated control of the analog electronics is required when both measurement modes are on. The analysis of requirements for domain engineering has a result in a features model, that has been structured in packages (Figure 5). The abstract features encapsulated in three main abstract domains MeasurementController, DataManagement and DataAcquisition, are completely reused in all the derived products. The AbstractSpectrometerFeatures package has the highest degree of reusability but also the highest degree of dependability. The abstract features depend on the commonality between EGY and SEC features. A change in the problem domain of one of the three products is mostly reflected in the degree of reusability of the abstract domain features.

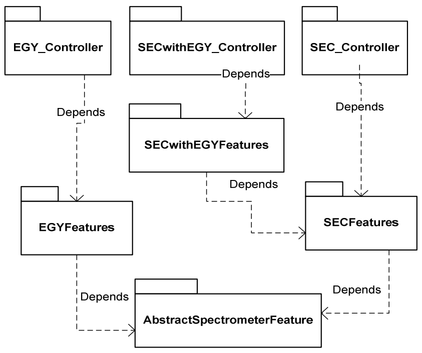

Figure 5: Mapping features into packages.

The sets of products that could be derived from the domain specific services during application engineering are: (1) P1 – EGYController includes specific services of a standalone control of EGY mode; (2) P2 – SECController includes specific services of a standalone control of SEC mode; (3) P3 – SECwithEGY Controller includes specific services of coordinated control.

The architecture model is documented around multiple views describing conceptual and concrete levels, for each view a static and dynamic perspective being offered. Architecture documentation addresses specific concerns for measurement control, data acquisition control and data management. The views are illustrated with diagrams expressed in UML-RT, a real-time extension of UML. The conceptual level considers a functional decomposition of the architecture into domains. The concrete level considers a more detailed functional description, where the main architectural elements are packages, capsules, ports, protocols. Conceptual View is the result of a functional-based decomposition, and it includes relations between different domains. The architectural components are large functional (domain) entities, and the connectors are *"uses", "command" or "passes-data-to"* relations. This structure is useful for understanding the interactions between entities in the problem space, for understanding the cross-domain perspective, and hence thereafter, the possibilities for creating a system of systems. It includes: (1) Measurement Controller Subsystem (MCS) which has the main role in controlling acquisition and dumping science data. (2) Housekeeping (HK) forms the reports and sends them to command interface when requested the command interface subsystem. It uses services provided by PMS. (3) Command Interface Subsystem (CIS) hides the hardware buses' interfaces from the rest of the software. (4) On-board clock (OBC) maintains an on-board clock used for time-stamping spectra in data files. It includes services for timing the start/stop of spectra and targets and other timing related services. (5). Memory Management Subsystem (MMS) provides services for handling the storages in RAM and EEPROM areas. (6) Parameter Management Subsystem (PMS) provides services for initiating, changing and reading the on-board parameters in EEPROM. (7) StartUp implements the power up and watchdog timer start-up. (8) Communication buffer management (BUFMAN) provides services for allocating/deallocating transmit buffers. (9) CPU specific services provides highly optimized high speed assembly language services (high speed word copy, interrupt enable/disable). (10) Hardware encapsulation modules control specific hardware (analog electronics, watchdog timer).

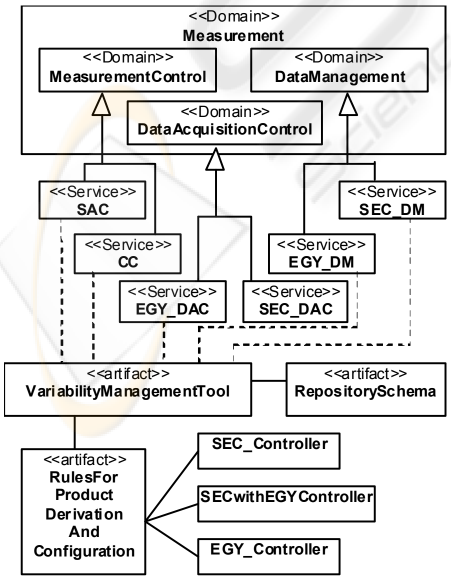

Figure 6: Measurement cross domain RA.

In a detailed functional decomposition view the main elements are packages, components, ports and protocols. The static relations between components are association, specialization, generalization, etc. Considering the dynamic relations, statechart diagrams and sequence diagrams are also part of this view. In this view abstract components are included based on a recursive control architecture style (Selic, 1998). The MeasurementControl domain includes services that are responsible for starting and stopping the operating mode for data acquisition according to the commands received from the command interface, and according to the events generated in other parts of the software. The DataAcquisitionControl domain includes services that collect events (science data) to the spectra data file during the observation of a target. This domain includes as well as hides data acquisition details. DataManagement domain includes services that provide interfaces for storing science data - opening/ closing/writing the data files, hiding data storing details. This domain also provides services for controlling the transmission of stored data to the command interface.

5 MODIFIABILITY ANALYSIS

One of the most important quality attribute for our system from a developer viewpoint is modifiability. Thus we have applied the analysis method introduced above for modifiability evaluation of the RA. We have defined change scenarios for different categories of changes. For each category we'll exemplify in the following with a scenario. The effect of the scenarios on the service components and the required views of the architecture are analyzed.

5.1 Change Scenarios Effects

1. General Purpose Hardware Changes Scenario. "Change the central processing unit (CPU)".

Effect on the architecture: CPU specific services provide highly optimized high speed assembly language services (high speed word copy, interrupt enable/disable, etc.) The services are not applicable at the level of description.

Result: Not applicable to the available views.

2. Domain-specific Hardware Changes Scenario. "Add a hard disk for SEC product".

Effect on the architecture: The SEC_controller and SECwithEGY_controller contain a hard disk for data storage. This scenario requires a lot at the architectural level, most of them related to the Data Management domain.

Result: Multiple changes in detailed functional decomposition, localized in the SEC_DM specific domain service.

3. Technology Changes Scenario. "Change the generator polynomial (different from CCITT polynomial) for 16 bit CRC sum of errors handlers".

Effect on the architecture: MMS consists of service functions for managing the storage RAM and EEPROM. It also includes a state for refreshing RAM and the memory error exceptions handlers (double and single bit).

Result: Modification in one component in the conceptual view.

4. Functional Requirements Changes Scenario. "How is the architecture affected when the operation mode is changed?"

Effect on the architecture: The operation modes is one of the variability among domains. This is encapsulated into DataAcquisition and DataFileManagement. The measurement control domain is decoupled from the operation mode of different products, which is encapsulated into the DataAcquisition domain.

Result: No change to the RA – abstract concrete or features of measurement control.

5. Non-functional Requirements Changes. "How is the average SRG-bus speed of 744kbit/sec on reading data from disk, which is time critical, maintained? "

Alternative solutions: (1) Change the hard disk: Use Fast disk: Optimal disk interleaving factor and storing the data file in sequential sectors on the disk. (2) Send filler blocks to the bus while waiting for the disk – a sufficient number of filler blocks could be reserved in the vector word sent in advance to BIUS. (3) Use a busy bit of SRG-bus. (4) Optimize disk driver – If the disk drive has been changed, the software has to be tuned separately for the new disk.

Result: Not applicable to the available views.

5.2 Scenarios Interaction

A good architecture design must provide a good localization of changes. Most of the changes required by scenarios were applied to one service component, which indicates a good decoupling of concerns. The most important change was the addition of the hard disk, a variability among domains. This scenario required changes to the domain specific services. By structuring the RA in abstract services, which encapsulate abstract features of the domains and concrete components, which in turn represents specialization of the variable features, the effects of the change scenarios are minimized and localized.

6 RELATED WORK

At this moment various architecture analysis methods, such as scenario-based architecture analysis (SAAM) (Kazman et al. 1996), architecture tradeoff analysis (ATAM) (Kazman et al., 1998), architecture level analysis of modifiability (ALMA)(Bengtsson et al., 2004), or software architecture reliability analysis using failure scenario (SARAH) (Tekinerdogan, 2008). Our study of the existing state-of-art research into the quality analysis methods reveals that the methods are distinguished by taking into account the evaluation techniques (qualitative or questioning, such as scenarios; quantitative or measuring, like metrics, etc.), the number of considered quality attributes and their interaction for tradeoff decisions, the stakeholders' involvement, and how detailed the architecture design is at the moment the method is applied to the architecture-based development process. Our method is based on the SAAM (Kazman et al., 1996) and ALMA, but improved through the introduction of guidelines for analysis. This is because the analysis is performed iteratively with design towards improvement. Another important novelty of our approach is that the method is specifically focused on a cross domain RA quality analysis that is on the first abstraction level of architecture development. Our main concerns are core services changes in the scenarios interaction step.

7 CONCLUSIONS

For the moment, only scenarios could be used in RA analysis for modifiability. One problem with scenario-based analysis is that the result and the expressiveness of the analysis are dependent on the selection of the scenarios and their relevance for identifying critical assumptions and weaknesses in the architecture. There is no fixed minimum number of scenarios whose evaluation guarantees that the analysis is meaningful. According to this, we tried to use five categories of possible changes in general hardware, specific hardware, functionality, non-functional requirements and software technology. A helpful strategy in scenario elicitation is the analysis of commonality and variability. This is not a part of the analysis method, but it is considered a pre-condition of it. One aim of the analysis should be to show how flexible a RA is in order to handle the anticipated changes provided by the variability of domains. Another aim is to analyze which is the potential of the RA to be adapted to changes in common features.

The results of the analysis depend not only on the views of the architecture, but also on the level of detail of the services descriptions. By using only the conceptual view the effects of the change scenarios are reduced. On the detailed functional decomposition view, which has been developed with the help of a CASE tool, the effect is more relevant. The interaction of unrelated scenarios is lower and it reveals a good separation of concerns when the domains decomposition is detailed.

ACKNOWLEDGEMENTS

This work is supported by the Romanian research grant CNCSIS IDEI no. 1238/2008.

REFERENCES

Bass L., P. Clements, R. Kazman. 1998, *Software Architecture in Practice*. Addison-Wesley.

Bengtsson PO., J. Bosch, 2004, Architecture Level Modifiability Analysis, Procs of the ICSR5.

Bosch J, 2000, Design and Use of Software Architectures - Adopting and Evolving a Product-Line Approach, Addison Wesley.

Buschmann F., R. Meunier, and H. Rohnert, 1996, Pattern-Oriented Software Architecture:A System of Patterns. John Wiley and Sons.

Dobrica L., E. Niemelä, 2002, A Survey on Software Architecture Analysis Methods, IEEE Trans on Soft. Eng, vol 28(7).

Dobrica L., E. Niemelä, 2008, Quality and Value Analysis Method for Product Line Architectures, Procs of ICSOFT 2008, 64-71.

Dobrica, L., Ovaska E., 2009, A service based approach for a cross domain reference architecture development, Procs of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 9-10 mai, 33-44, 2009, INSTICC Press.

Gamma E., R. Helm, R. Johnson, and J. Vlissides, 1994, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley.

Graaf B, vanDijk H, van Deursen A., 2005, Evaluating an embedded software reference architecture – industrial experience report, CSMR 2005, 354-363.

Kazman R., G. Abowd, L. Bass, P. Clements, 1996 *Scenario-based Analysis of Software Architecture*. IEEE Software, pp. 47-55.

Kazman R., S. J. Carriere and S. G. Woods, 2000, Toward a Discipline of Scenario-Based Architectural Engineering, *Annals of Software Engineering*, Vol. 9.

Selic B., Recursive control, in: R. Martin, et al. (Eds.), Patterns Languages of Program Design—3, Addison-Wesley, 1998, pp. 147–162.

B., Sozer H., and M. Aksit. 2008, Software Architecture Reliability Analysis using Failure Scenarios; Journal of Systems and Software, 81, 558-575.