Experiences in Making Architectural Decisions during the Development of a New Base Station Platform

Juha Savolainen[1], Juha Kuusela[2], Tomi Männistö[3], and Aki Nyyssönen[4]

[1] Nokia Research Center, Itämerenkatu 11-13, 00180 Helsinki, Finland
juha.e.savolainen@nokia.com
[2] Nokia Devices, Keilalahdentie 4, 02150 Espoo, Finland
juha.kuusela@nokia.com
[3] Aalto University, Tekniikantie 14, 02150 Espoo, Finland
tomi.mannisto@tkk.fi
[4] Nokia Siemens Networks, Kaapelitie 4, 90650 Oulu, Finland
aki.nyyssonen@nsn.com

Abstract. Creating architecture for a complex telecommunication system is a difficult task and requires expertise of many different stakeholders. The software architecture design process relies on understanding the architecturally significant requirements (ASRs) for the system under design. This paper describes experiences in creating a new base station product line. A goal was to create a process to facilitate fulfillment of ASRs during the development of the product line. The approach proved to be feasible for developing large-scale systems in the telecommunications infrastructure domain. This paper describes the approach taken, experiences gathered during the development process and promotes the idea of defining concrete ASRs for each project and refining them through architecture for all relevant subsystems.

Keywords: Software architecture, design decisions, architecturally significant requirements process, refinement.

1 Introduction

Software architecture has been identified as one of the main development tools that helps system to achieve it's quality requirements [1]. Correct architectural decisions allow the system to meet its quality requirements such as modifiability, performance, reliability and security. In the highest level, architectural styles has been proposed as the way to match architecturally significant requirements to architectural decisions [2]. However, architectural styles cover only the highest levels of software architecture design. In fact, it seems that complex systems tend to be based on many different architectural styles.

The overall system architecture determines the main subsystems. Separation to the subsystems can happen for many reasons, but for large-scale systems the most important is that the subsystems support division of work. That is, each subsystem can be reasonably independently designed and implemented. This is measured by cohesion and coupling. By increasing the cohesion within the subsystem and minimizing coupling between the subsystems can make the subsystems more independent.

In this paper, we share experiences from a project in the telecommunication domain. The intention of the project was to build a new base station platform that would allow creating base stations for various telecommunication standards. The base station project was substantial in size. More than one hundred engineers and multiple sites around the globe participated contributed to the project. Because of the geographical division, a way to communicate the decisions and to effectively divide work to the different sites was very important.

In the context of our work, the system had chief architects that were responsible for the whole system and subsystem architects for the system components, the first level of the decomposition. Together these architects intended to create an architecture that would supports ASRs for the whole system. Initial list of ASRs was created by the chief architects together with architecture experts from corporate research. The highest-level software architecture document was created to describe how the ASRs were addressed by the overall architecture.

For each of the system components the system-wide ASRs must be converted to the ones that are relevant to that particular subsystem. As the design progresses each system component will determine its own architecture within the constraints of the higher-level architecture and will again will define new subsystems. We consider architecture design to continue until no new subsystems are needed. At this point the design will continue with classes or code modules, depending on the implementation technology.

The structure of the development project was aligned with the software architecture of the system. This provided easy communication of architectural decisions through software architecture documentation.

The remainder of this paper is organized as follow. Section 2, discusses three main ways in which we made architectural decisions during the base station project. After this, we discuss our findings and conclude in Section 3.

2 Observations on Architectural Decisions during the Development of the Base Station Platform

Designing good architectures require making correct design decisions in many different levels of abstraction. On the higher levels many questions relating to overall structure of the architecture must be answered, general approach for division of work established and main responsibilities of software elements decided. On the lower levels APIs must be designed, process structures decided and detailed compile time dependencies managed.

A number of researchers have argued that architectural decisions should be, if possible, delayed. Delaying decisions achieves flexibility and allows improved knowledge taken into account when making the decisions. Lower level decisions are equally difficult to be taken upfront, since not enough is known on the implementation details to make correct decision on e.g. detailed process or threading design issues. We agree with Ruth Malan and Dana Bredemeyer [3] who argue that an architect should make as few decisions as possible, deferring the rest until later in the lifecycle. Delaying decision means that decisions must be made during different time in the system development lifecycle and in a different level of abstraction.

In the base station project, we had three main ways to target architectural concerns. Sometimes an architectural solution was created and described in form of components, their dependencies and interactions, allocation of functionality or interfaces. Often, chief architects could decide to solve the issue later. This was communicated to subsystem architects using system component specific ASRs. In few cases, an additional process step was sometimes added to alleviate the risk of not satisfying the concern at some later point of time.

We believe that understanding the different intent of these different approaches to architectural decisions will help architects to better express their decisions in their own architectures.

2.1 Architectural Decision Are Made in Form of the Structure

The most common type of architectural decision is a decision taken to modify the current structure of the software architecture. These decisions affect the architecture on level of abstraction currently under work and typically address the ASRs defined for this level of abstraction. In general, the structure is a good way to communicate decisions. When a decision is expressed as e.g. a defined interface or as a responsibility division between subsystems it makes the decision very explicit and the probability of misunderstandings is low. The decision can be implemented by updating the structure.

For our project, the intent was to use the overall list of ASRs as the way to organize architectural decisions. The main ASRs were included into all versions of the architectural documentation to keep track on the process of facilitating them.

Not all decisions affecting the structure give ready-made solutions to ASRs. Some architectural decisions are made as explicit constraints on future decisions. These are often explicitly documented as constraints or rules [4], but can be sometimes disguised as just another architectural decision on the structure of the architecture.

An architectural constraint restricts the future architectural decisions. Even though it can be argued that all decisions will reduce the available design space [5], for architectural constraints this becomes the essence of the decision. One possible way to express a constraint is to use layer diagrams as shown in Figure 1.

The idea of layer diagram in Figure 1 was to communicate to subsystem architects and other stakeholders the main mechanism to achieve reuse in software product line architecture. Layers were used to achieve reuse across different radio access technologies, that is, to allow derivation of products that differ on the supported radio technology. Hence, two layers were constructed, one specific to radio technology and one generic.

Most subsystems could be mapped to either layer, but some had functionality that belonged to both layers. For these subsystems, the subsystem architects must guarantee that on the component level, the subsystem can be organized to a layered structure that separates components that are specific to a particular radio technology and to those that contain generic functionality.

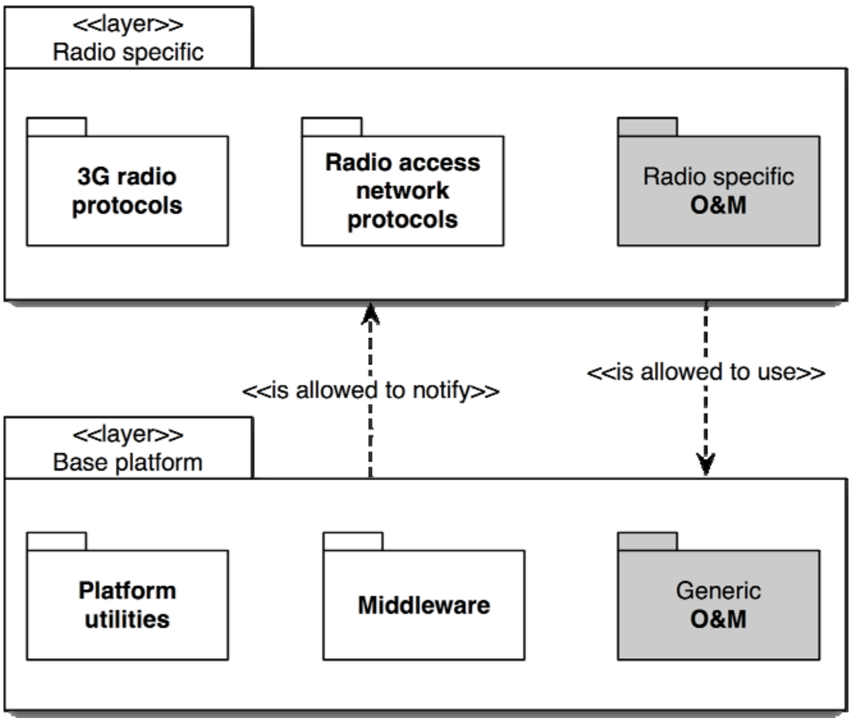

Fig. 1. Architectural constraints shown in the context of the structure

2.2 Architectural Decisions Are Done in Form of Requirements to the Structure

Making architectural decisions in the form of the structure requires that we can describe the decision in terms of architectural structures. Sometimes, the architects either lack the information, necessary specialized expertise, or some preliminary decisions have not been done so that it is not possible to express the decision in terms of structure.

Decision has to be expressed in form of architectural requirements. Here the intention is not to make the actual decision, but the explain the requirements for the decision to be made later. A requirement places a constraint that is described in a textual form and targets some subsystems (or system components). This way of communicating decisions is typically less precise than making the decision in form of the structure, but is necessary when the architects cannot make the actual decisions right now and must delay them.

However, the decisions on the structure are not independent from decisions in form of requirements. In particular, an architectural decision on the level of the entire system can greatly affect the ASRs for the subsystems. For example, a decision to distribute responsibilities between subsystems defines what ASRs will be relevant for each subsystem. That is, making decisions on the structure of the architecture and making the ASRs are results of an iterative process of architecting and both communicate architectural constraints.

As long as architecture design continues, we believe that it is essential to refine the ASRs, so that for each step of software architecture design, the relevant ASRs are available. To do this, two things must happen. First, for each level of decomposition a subsystem architect must be able to identify the relevant higher level ASRs for it and second, the architect must be able to derive the relevant ASRs into subsystem specific ASR(s). This process is described in Figure 2.

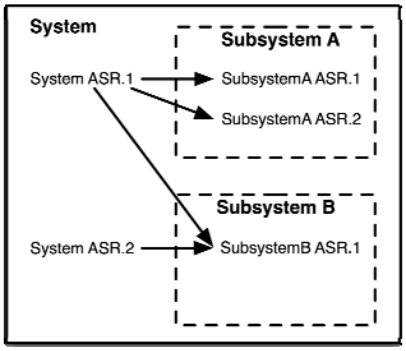

Fig. 2. Refining ASRs in general (adapted from [6])

Figure 2 describes how ASR refinement works in general. Two system-wide ASRs refined to the subsystem ASRs. There is a complex mapping between the system ASRs and the subsystem ASRs. Often a system level ASR is refined into many subsystem level ASRs similarly what ASR.1 is refined into subsystem A ASRs SsA_ASR.1 and SsA_ASR.2. This represents how higher-level ASRs become more detailed when transitioning to subsystems. Also many system level ASRs affect a number of subsystems. In Figure 2, the system level ASR.1 if refined to ASRs of both subsystems A and B.

However, not every ASR needs to be relevant to all subsystems. For example, performance may be crucial for data transmission part of the system, but a maintenance system can ignore strict real-time requirements. In Figure 2 this is shown in case of ASR.2 that is only refined into respective ASRs for subsystem B.

For the base station platform we had an extensive excel document describing how the system level ASRs were reflected in the subsystems. The idea was not to repeat the ASRs in the subsystem documentation in the same way as they were described in the overall architecture document, but rather describe the ASR as requirements for the subsystem. We felt that is was crucial to express the subsystem ASRs in those terms that are relevant for the subsystem. The resulted subsystem ASRs were then recorded and represented in the subsystem architecture document.

For example, a system level ASR "The system shall be able to report any fault in the system within 5 seconds of its detection" could be refined to BTS O&M system component ASR as "BTS O&M shall report a detected fault in any of the HW cards within 2 seconds". This means that we intended to make the ASRs always very relevant for the particular subsystem and also communicate e.g. performance budgets.

After defining some of the subsystem level ASRs will start another iteration where decisions on the (subsystem) structure will be made and new detailed ASRs for the components within the subsystems are born.

2.3 Architectural Decision as a Process Step

In two previous sections we have discussed two ways to address software architecture development directly by either making architectural decisions in form of the structure or as architectural requirements. However, during the software architecture design, not all decisions address the architecture directly, but affect the process of designing the architecture.

For a large system, the chief architect is typically responsible for the overall system architecture and each subsystem has an own responsible architect. A set of rules cover the whole architecting process and give guidelines on what documentation is required for each subsystem. These rules are often derived from company or domain specific standards on architectural documentation.

Based on the decisions on how to achieve ASRs in the system level, the subsystem ASRs are affected. Besides this, also the ways the architectural decisions are documented or intended to be pursued, could be tailored for some subsystems. This is the essence of architectural decisions as a process step. For example, some subsystems may be required to document how certain aspects of the system are achieved or defined analysis may be required.

In the base station, two large decisions on the process of architecting were done. First, for selected subsystems architectural evaluation was decided to be done in a later stage to verify the fulfilment of a number of key ASRs. Second, a detailed analysis of performance characteristics was selected for a small subset of the architecture. This part of the architecture was agreed to be critical for the overall performance of the messaging functionality of the base station. We decided to perform rate monotonic analysis (see e.g. [7]) for this part of the system after we have better understanding on the actual process structure. This process decision was combined with architectural requirements, because we also decided to estimate worst-case execution times for each system component that limited the design freedom of the system component teams.

3 Discussion and Conclusions

Software architecture can be been seen as a series of architecturally significant design decisions. These decisions aim to define the architecture that is then used as the basis for system implementation.

The decisions can manifest themselves as being about requirements, architecture or changes to design process, among others. The kinds of decisions include, for example, requirements, constraints, rules and different decisions about the actual systems, such as structure, interfaces, styles, patterns and detailed design decisions. These all can be seen as restrictions on the allowed design space in the later phases of the design process.

In a broader view, we try to shed light in this paper on the issue that when someone is making architectural decisions, what kind of decisions should she make. There are two main aspects in that. First, the decisions should be made to the appropriate level of detail not to restrict the later design decisions unnecessarily. Second, the decision-making should bear adequate responsibility not to postpone the making of decisions on issues that could and should be resolved here and now. Within these boundaries, the decision-maker should be aware of her capabilities in making the decision, so that she would have the a justified understanding on how concrete decisions to make or what issues to still leave to be determined later, typically by others.

We used a concrete example of the phenomenon in which the ASRs of the whole system were propagated to the subsystems. The propagation was not straightforward and thus illustrated how ASRs for the whole system become refined and take a different form when interpreted within the context of a subsystem and how in this decision-making the structural decision regarding the system decomposition have a major impact on the ASRs of the subsystems.

The used case example is a real example from a complex architecture design problem. However, the example is simple in the broader view of the idea we aim to illustrate, as it only covers a slice of the entire problem area in making different kinds of architectural decisions. There is clearly a room for deeper understanding on the kinds of architectural design decisions and their role in restricting the available design space. With such understanding, a framework and guidelines could be defined to help practitioners to explicate their own boundaries and responsibilities and to assess what kinds of architectural decisions should be applied in a particular situation.

For practitioners it is important to make explicit decisions on how to represent architectural decisions. If the architect's have enough knowledge to make the decision by affecting the structure of the architecture and this simplifies the further development by supporting independent work, then the decision should be made. Otherwise creating new subsystem specific ASRs or adding a process step to e.g. evaluate subsystem architectures may be more appropriate. Despite that we described three ways to make architectural decisions separately; these three options are rarely independent. When making a decision on structure, it tends to change subsystem-level ASRs. In addition, the more decisions are made; typically less process is needed later.

The ability to make correct architectural decisions and how to represent them is ultimately dependent on the competence of the architect. This paper presented three different ways to represent architectural decisions to assist architects to consider all options during the architecting process. In the future, we intend to research more on factors affecting the choice of representing architectural decisions in order to further assist practitioners.

References

[1] Bass, L., Clements, P., Kazman, R.: Software architecture in practice. Addison-Wesley, Reading (2003)

[2] Shaw, M., Garlan, D.: Software architecture: Perspectives on an emerging discipline. Prentice Hall, Englewood Cliffs (1996)

[3] Malan, R., Bredemeyer, D.: Less is more with minimalist architecture. IEEE IT Professional 4(5), 46–48 (2002)

[4] Bosch, J.: Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. Addison-Wesley, Harlow (2000)

[5] Davis, A.: Great Software Depates. Wiley, Hoboken (2004)

[6] Savolainen, J., Kuusela, J.: Transition to Agile Development - Rediscovery of Important Requirements Engineering Practices. In: 18th IEEE International Requirements Engineering Conference. IEEE, Sydney (to appear, 2010)

[7] Nord, R.L., Cheng, B.C.: Using RMA for evaluating design decisions. In: Proceedings of the IEEE Workshop on Real-Time Applications, pp. 76–80 (1994)