Requirements for Service Architecture Modeling

Mari Matinlassi, Jarmo Kalaoja

VTT Electronics, P.O Box 1100, 90571 Oulu, Finland
{Mari.Matinlassi, Jarmo Kalaoja}@vtt.fi

Abstract. The development of large and complex software intensive systems, among other issues, means various stakeholders related to software architecture modeling. Various stakeholders concern different aspects of software development, e.g. management, testing, component engineering or product marketing. Therefore, fluent communication between various stakeholders may become a difficult and confusing issue in software development. This paper introduces the requirements for service architecture modeling. The requirements include definitions of four viewpoints at two levels of abstraction. The rationale for two levels and four viewpoints required in service architecture modeling are introduced first. In addition, this paper also defines the viewpoints by means of concerns, stakeholders and artifacts. This paper also maps the service architecture description approach to the OMG's model driven architecture approach. The service architecture modeling approach introduced here improves communication among different stakeholders and increases reusability of higher-level architecture descriptions.

Keywords. Software architecture, architecture description, service architecture, viewpoint, Platform Independent Model (PIM), Platform Specific Model (PSM)

1 Introduction

The model driven architecture (ISO/IEC 10746-3:1996 2001) defines a normative model that guides the specification of IT systems. The main idea is to separate the *functionality* descriptions from the *implementation* specifications and therefore increase the integrability during system evolution. That is, functionality descriptions independent from the implementation platform last for a long time, while implementation technologies are 'the thing' just for a while and will become obsolete sooner or later. This idea of separating implementation-specific issues from functionality descriptions equals the notion of *software architecture*. The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them (Bass et. al 1998). Software architecture also includes principles that guide the design and evolution of architecture (IEEE Std-1471-2000 2000). Although implementation independence is not literally included in the definition of software architecture, implementation independence is still there. To be more accurate, software architecture is independent of implementation *language*. In addition to those definitions above, software architecture also has a wider meaning. The taxonomy of the formally defined orthogonal properties of software architectures, i.e. TOPSA (Bratthall & Runeson 1998), extends the definition of architecture. It defines a space with three dimensions: abstraction (conceptual, realization), dynamism (static, dynamic) and aggregation.

Accordingly, we suggest a *conceptual* architecture description, which identifies software architecture in terms of abstract criteria rather than realization ones, whereas *concrete* architecture description captures architectural issues closer to software realization. In addition, both architectural descriptions need several viewpoints in order to represent the whole system from various perspectives e.g. structure or behavior. Every viewpoint of conceptual architecture is an *abstraction* of the ones in concrete architecture. Abstraction means the selected removal of information i.e. bigger components, fewer details and deferred functions. Conceptual architecture descriptions are essential in the early phases of design when roughing out the structures of software.

Here, architecture modeling is considered for the domain of service architectures in particular. A service is the capability of an entity (the server) to perform, upon the request of another entity (the client), an act that can be perceived and exploited by the client (Niemelä et. al 2002), whereas service architecture is the architecture of applications and middleware. It is a set of concepts and principles for the specification, design, implementation and management of software services (TINA Consortium 1997).

This paper introduces the background for two levels and four viewpoints required in service architecture modeling, and it also introduces the viewpoints by means of concerns, stakeholders and artifacts. This paper also maps this service architecture description approach to the OMG's model-driven architecture approach.

2 Architecture Modeling Requirements

2.1 Background

When designing software architectures it is not feasible to begin with the bottom-up approach, because one is expected to consider the system in detail. Instead, one needs to use a top-down approach to the issue. A conflicting practice within architectural documentation today is that top-level architectural descriptions are not supported. Lower level documentation does not reflect all the thoughts the architect had in mind in the early phases of the design. Documentation is important because, in most cases, the adapters of the architecture, e.g. software integrators or administrators, are not its creators. With high-level architectural descriptions available, it is easier for the adapters of the architecture to use a top-down approach when familiarizing themselves with the structures and activities in a system.

Software architecture has to be described so that various stakeholders, e.g. customers, management and developers, understand it. A common understanding is difficult to achieve, because even if the basic intention of stakeholders is similar, different stakeholders need information at different levels of abstraction and aggregation. Because of the above reasons, the service architecture description needs to be divided into *conceptual software architecture* and *concrete software architecture* (Matinlassi et. al 2002).

The stakeholders in service architectures and their descriptions are introduced in Table 1. These various stakeholders use the service architecture descriptions for the following purposes:

- acquiring an overview of available services and their use,
- describing the responsibilities and context of components,
- allocating and understanding the division of work,
- mapping services to components and vice versa,
- mapping specific services to generic services,
- clustering the components to be developed into technology domains,
- considering the appropriateness of service architecture,
- finding out what quality issues are considered,
- tracing how attempts are made to achieve qualities with architectural styles and patterns and why these qualities are important; and
- understanding and integrating third party components

Table 1. Service architecture engineering stakeholders

Category	**Stakeholder**	**Description**
Services	System architect	Develops a system architecture, Hw/Sw partitioning
	Service user	Uses services defined by the service architecture
	Service provider	Provides services for service users
	Service developer	Develops services for service providers
Components	Component designer	Designs components that provide services
	Component integrator	Integrates available components into services
	Component developer	Designs, implements and tests software components
Products	Product architect	Creates a product architecture
	Product developer	Develops product specific part of software, integrates components
	Product marketing	Presenting product (variable) features to customers
Software	Manager, assets manager, reuse manager	Manages, deals with costs and benefits, business, technology and reuse strategies
	Software architect	Develops software product (line) architecture
	Testing engineer	Tests software packages, integration testing
	Maintainer	Upgrades products/systems

Because of the reasons mentioned above, it is obvious that one kind of architectural description is not enough, but that the architecture has to be described with several different views. We now turn to consider the different viewpoints required in service architecture modeling.

2.2 Service Architecture Modeling Viewpoints

According to IEEE Std-1471-2000 (2000), an architectural view is a representation of a whole system from the perspective of a related set of concerns. In the literature, there are several approaches to the design of software architecture that concentrate on different views of architecture. The first of these view-oriented design approaches was the 4+1 approach, developed by Krutchen (Krutchen 1995). After this, several others have approached the jungle of architectural viewpoints. For instance, Jaaksi et al. introduced their 3+1 method in 1999 (Jaaksi et. al 1999) and Hofmeister et al. used all in four views to describe architecture (Hofmeister 2000). Among these approaches there is no agreement on a common set of views or on ways to describe the architectural documentation. This disagreement arises from the fact that the need for different architectural views and architectural documents is dependent on the two issues: system size and software domain e.g. the application domain, middleware service domain and infrastructure service domain. Again, both the system size and domain have an impact on the amount of different stakeholders. Therefore, it is obvious that none of these methods alone is comprehensive enough to cover the design of software architectures for systems of a different size on various domains, or provide an explicit means to create architectural descriptions for all the systems.

We are neither trying to cover all the domains nor to define a catchall set of architectural viewpoints. Instead, we concentrate on the service architecture domain and the viewpoints needed in service architecture modeling. This paper is based on the three viewpoint elements defined in (Matinlassi et. al 2002). It extends the modeling requirements with a definition of the fourth viewpoint, the development viewpoint. Viewpoints for both levels of abstraction are similarly named: structural, behavior, deployment and development (Figure 1). The viewpoint extension is based on the experimentation in service architecture case studies in the WISE project[1] and in a national joint research project with industrial case studies.

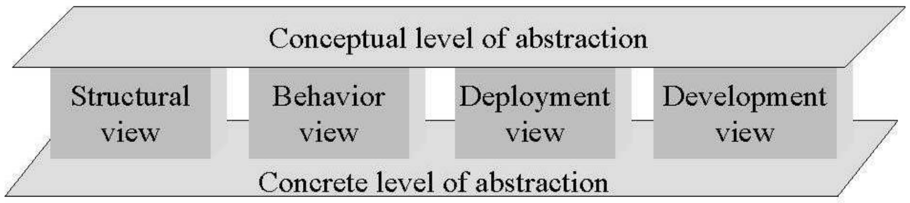
Conceptual level of abstraction
Structural view
Behavior view
Deployment view
Development view
Concrete level of abstraction

Figure 1. Views at two levels of abstraction in service architecture modeling.

Fragments of the viewpoint elements are shown in Table 2 and Table 3 (Purhonen et. al 2002). The tables capture the issues each view *concerns*. These issues with which each view is concerned are aimed at certain *stakeholders*. Each view also produces its own specific *artifacts* i.e. models or diagrams that provide appropriate information for the stakeholders. The differences between two levels of abstraction lie in the following issues. First, in the degree of details expressed by the architecture and partly also in the depth of aggregation levels.

[1] A European project, IST-2000-30028.

Table 2. Summary of the elements of conceptual service architecture

Name	**Conceptual structure**	**Conceptual behavior**
Concerns	What services and components are required, what are the responsibilities of services, how are quality requirements met?	What kinds of actions does the service architecture provide for applications, which services collaborate in each action, how are actions related to each other?
Stakeholders	System architects, service developers, product architects and developers, maintainers	System architect, component designers, service developers
Artifacts	Table of clustered functional responsibilities, table of quality attributes, decomposition model	Table of interaction scenarios with services, interaction model
Name	**Conceptual deployment**	**Conceptual development**
Concerns	Which kinds of nodes are there in a system, what services have to be in the same unit of deployment, how can services be allocated to nodes?	What services and components does the company develop and what services are acquired from third parties, who is responsible for a service, which standards and enabling technologies do the services use?
Stakeholders	Service users, service developers	Project manager, component acquisition
Artifacts	Table of units of deployment, node definition, allocation model	An acquisition model, a constraints model

Table 3. Summary of the elements of concrete service architecture

Name	**Concrete structure**	**Concrete behavior**
Concerns	What are the concrete components needed for a corresponding conceptual component, what are the interfaces needed, how do services communicate with external actors?	How does a concrete component behave and respond to an event, what is the behavior of a set of concrete components?
Stakeholders	Component designers, service developer, product developers	Component designers, testing engineers, integrators
Artefacts	Hierarchical structure diagrams, fine-grain context diagram	State diagrams, message sequence diagram
Name	**Concrete deployment**	**Concrete development**
Concerns	What nodes and devices are there in a system and what do they have to do, what concrete components are allocated to each node and device?	What is the realization of a service or a component, how does a service or a set of services relate to each other, how could a service be configured?
Stakeholders	Integrators, maintainers	Product developers, assets managers
Artefacts	Deployment diagram	Table of component realisations, configuration rules, links to the assets repository

3 Mapping to MDA

In this section, our aim is to elucidate how and to what extent our approach conforms to the model-driven architecture approach. It is done by comparing the concepts and notions related to our approach and to MDA.

First, the model driven architecture defines that a *platform* refers to the implementation details of a software component, wherein these technological and engineering details are irrelevant to the fundamental functionality. However, the platform is not necessarily implementation *language* environment-specific. Therefore, the platform may equal e.g. a component model. Our approach defines *conceptual* architecture that is at a higher level of abstraction.

Therefore, conceptual architecture is not a platform-specific model, whereas *concrete* architecture may comprise platform-specific details that are implementation language-independent.

Again, by *model* in model driven architecture (MDA), it is meant a *formal* specification of a part of a function, structure and/or behavior of a system. Formal specification expects either textual or graphical language with defined syntax (i.e. notation) and semantics (i.e. meaning). The approach, we propose here, does not define a strict formal language. However, we have used an experimental notation in (Matinlassi et. al 2002) and a refined, more consistent notation based on UML, for both the conceptual and concrete levels is under work. If comparing the experimental notation and its refinement it can be concluded that the architectural elements and the way that they are used remains the same, regardless of the changed graphical expressions.

In addition to the use of different viewpoints, the MDA encourages the use of different levels of abstraction (ISO/IEC 10746-3:1996, 2001). This refers to zooming in/out of objects and/or interactions. Zooming out in practice means getting a simplified model with fewer details, whereas zooming in is seeing those details. In addition, a not so flexible decomposition and composition are also included in the MDA. Our approach realizes the zooming function through two abstraction levels in architecture descriptions. Decomposition, i.e. aggregation dimension in architecture space, is also implemented in our approach at both levels of abstraction (conceptual and concrete).

A platform-independent model (PIM) is a formal specification of the structure and function of a system. This specification abstracts away the technical detail, whereas the platform-specific model (PSM) is a model bound to a specific platform (e.g. to CORBA or SOAP). Furthermore, according to the ISO/IEC 10746-3:1996 (2001), "MDA defines *an architecture for models* that provides a set of guidelines for structuring specifications expressed as models", whereas our approach defines requirements for *software architecture descriptions* that are especially used in service architecture engineering. With the reasoning above, we can conclude that our approach corresponds to the MDA Platform Independent Model (PIM) and Platform Specific Model (PSM) as shown in Figure 2.

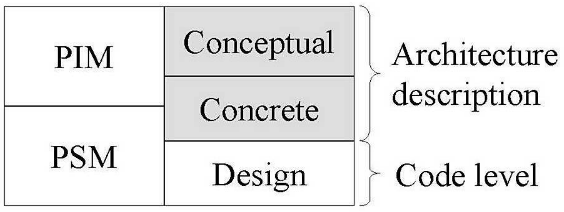

Figure 2. Mapping two level architecture descriptions to MDA.

As can be seen, the conformance is not perfect when considering the 'interface' between platform-independent and, on the other hand, platform-specific models. The conceptual architecture description includes only platform-independent issues, whereas concrete architecture considers platform-independent issues but also touches the implementation-specific details. Implementation-specific details are considered in the development view. That is, the point of development view is to consider e.g. how does the selection of a specific implementation platform influence the structures of concrete architecture, and is it possible to create concrete architecture that is platform independent, i.e. architecture that supports more than one implementation platform. The platform independence of concrete architecture can be further assisted by generative tools, which automate the transition from the architectural model to a spesific platform. These generative tools often implement similar mapping than that from PIM to PSM in MDA, however the tools transform models directly to code level, i.e. no platform spesific architectural model is used as an intermediate phase. This can provide signifigant savings in labor needed. The drawback is that this may lead to greater dependency on the CASE tool vendor than the approach used in MDA, because generic generative tools usable in real projects do not yet exist.

In an architectural design, it is not always reasonable to 'draw a line' between platform-independent and specific issues because it is not so obvious where and what the difference is. Therefore, it is simpler not to explicitly separate platform-specific architecture into a separate model. In our approach a separate view of architecture, development view is used to attack this problem in a flexible way in such cases. One important issue to consider is the reuse of COTS components. Furthermore, in architecture creation, many implementation (platform)-specific details are also mainly an issue of design level choices. The MDA is usable even then, but should in that case be considered more as a design level than an architectural approach. To conclude, the separation of concerns defined in the MDA approach is logical and straightforward but not always effortless or reasonable to be performed in architectural design.

4 Conclusion

The aim of this paper was first to give reasons for the necessity of two separate levels of abstraction and the need for multiple viewpoints in architectural representations. To sum up, the needs were argued using the varying needs of different stakeholders and business roles related to service architectures. The fact that conceptual architecture is required in order to improve common understanding among stakeholders that are all not necessarily software professional was also highlighted.

Furthermore, the proposed four viewpoints were introduced by means of concerns, stakeholders and artifacts. The views are structural, behavior, deployment and development. The conceptual structural view concerns the following issues: What services and components are required, what are the responsibilities of services and how are quality requirements met. On the other hand, the concrete structural view refines these issues with concrete component and interface definitions. Similarly, conceptual behavior deals with high-level design choices such as what kind of actions does the service architecture provide for applications, which services collaborate in each action, and how are actions related to each other. The concrete behavior view concentrates as well on concrete component behavior and how it responds to an event, as on the behavior of a set of concrete components. Deployment views concern service or component allocation to physical computing nodes at two levels of abstraction, including mutual requirement relationships between services. Furthermore, the development viewpoint participates in architecture description by providing a useful viewpoint on architecture development issues, such as the degree of reuse in components, work allocation and standard or technology issues as well as service configuration.

In the end, we discussed on how our approach conforms to the MDA approach. Through comparison of the concepts used in the approach we proposed here, and the concepts used in MDA, we concluded that the four viewpoints at two levels of abstraction mostly conform to the model driven architecture. This means that conformance includes a restriction that the interface between a platform independent model (PIM) and a platform specific model (PSM) in architectural design cannot always be strictly separated and platform specific modeling can be seen as a design level issue, and not necessarily an architectural problem.

References

Bass, L., Clement, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, Reading MA (1998)

Bratthall, L., Runeson, P.: A Taxonomy of Orthogonal Properties of Software Architectures. In: Proceedings of NOSA’99, University of Karlskrona (1998)

Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison-Wesley Longman Inc., Reading (2000)

IEEE Std-1471-2000, (2000) IEEE Recommended Practice for Architectural Descriptions of Software-Intensive Systems, IEEE Computer Society, 29p.

Jaaksi, A., Aalto, J-M ., Aalto, A. & Vättö, K. (1999) Tried & True Object Development. Industry-Proven Approaches with UML. Cambridge University Press, New York, 315 p.

Krutchen, P.B. (1995) The 4+1 View Model of Architecture. IEEE Software 12, pp. 42 - 50.

Matinlassi, M., Niemelä, E., Dobrica L.: Quality-driven architecture design and quality analysis method, A revolutionary initiation approach to a product line architecture. VTT Publications 456, Technical Research Centre of Finland, Espoo (2002)

Niemelä, E., Honka, H., Jormakka, H., Kalaoja, J., Koivisto, J. Kyntäjä, T., Latvakoski, J., Näyhä, T., Rannanjärvi, L., Valtanen, K., Vakivuo, T. 2002 Services architectures. In: Communications Technologies. The VTT Roadmaps. Sipilä, M. (ed.), VTT Research Notes 2146. Espoo 2002. pp. 45-60.

Purhonen, A., Niemelä, E., Matinlassi, M. 2002. Views of DSP Software and Service Architectures. Submitted to Journal of Systems & Software. 30 p.

ISO/IEC 10746-3:1996, (2001), Model Driven Architecture (MDA), OMG Architecture Board, 31 p.

TINA Consortium: (1997) Service Architecture Specification. http://www.tinac.com (2.7.2002)