SOA Process Decisions: New Challenges in Architectural Knowledge Modeling

Qing Gu
Department of Computer Science
VU University Amsterdam, The Netherlands
qinggu@cs.vu.nl

Patricia Lago
Department of Computer Science
VU University Amsterdam, The Netherlands
patricia@cs.vu.nl

ABSTRACT

Architectural design decisions are commonly agreed as one of the main elements that constitute architectural knowledge. To avoid knowledge vaporization, architectural decisions and their rationale need to be documented in a systematic manner. Various works have been done on representing architectural knowledge. Process decisions are one type of design decisions that existing models have paid little attention to. In this paper, we specifically address process decisions in the SOA domain. In doing so, we challenge architectural knowledge representations by mapping two SOA process decisions against a core model of architectural knowledge. The result suggests that existing models need to be extended in order to model SOA process decisions.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Design

Keywords

Architectural knowledge, model, process decisions, service-oriented architectures.

1. INTRODUCTION

Software architecture has been gaining great importance in the software engineering field [1]. Recently, software architecture is regarded as a set of architectural decisions [12]. These decisions, as well as architecture design, assumptions and linkage to the environment constitute architecture knowledge, which became a promising research topic in the area of software architecture [9, 24, 25]. Explicitly storing architectural knowledge in a systematic manner enables the interaction between stakeholders and increases the traceability of the architecture design. This way, knowledge vaporization is prevented [5].

SHARK'08, May 13, 2008, Leipzig, Germany.

During the software architecture process, choices (the design decisions) are made when contextual and technical knowledge (the rationale) is used to examine the tradeoff between requirements and solutions. Architectural design decisions and their rationale are regarded as the key elements of architectural knowledge [15].

There are different kinds of architectural design decisions. Some decisions are made by regarding the architecture as a product, such as structural decisions. For instance, choosing 3-tier architecture to increase the maintainability of the system or using redundant hardware to achieve desired availability is an example that directly relates to the architecture product. Some other decisions instead consider the development *process* as a product, i.e. they are *process decisions* [15]. For instance, choosing the waterfall software development process model is a decision that does not directly relate to the architecture design, rather to the development process that leads to the resulting architecture design. In this paper, we specifically address process decisions.

Recently, various architectural knowledge models [24, 15, 26] have been proposed for representing design decisions. Most of the architectural knowledge that is generated during the software architecture process is well represented by using the notations that the software engineering models entail. All of them are implicitly addressing those design decisions that regard software architecture as a product. Process decisions, however, are paid little attention to.

In traditional software engineering, software development processes have reached a high level of maturity. For instance, standard models such as the waterfall model or the spiral model serve as recognized patterns for software developers. When these models are used, software developers know exactly how to carry out development processes. Process decisions are often not modeled because they are not as significant as those design decisions on software architecture.

Such standard models do not yet exist in new paradigms like service engineering. Efforts still need to be put on systematic development methodologies. Consequently, process decisions play here a more important role than in traditional software engineering.

Service oriented computing (delivering software functionalities as services) is an emerging approach that is gradually becoming more popular and dominant in modern software engineering. Services become the building blocks of software systems. Nowadays, organizations often approach SOA in two ways. A less innovative way uses SOA as base for re-engineering, wrapping or migrating existing cooperate software systems towards SOA. In this approach, their IT port-

folio is migrated to the service landscape and used within the organization. A more innovative way to use SOA (and more auspicated) is delivering software as services where the customers of these services no longer request to buy and install applications; instead, they pay for the services usage. In this paper, we concentrate on the latter type of use. Consequently, new characteristics are introduced by the service oriented paradigm. These characteristics directly lead to a number of process decisions that have to be made in service oriented development and bring new challenges.

This paper addresses two of those new characteristics, namely the shift of the ownership of software and the cross-organization cooperation. Each of these two characteristics is discussed by giving an example of SOA process decisions elicited from the service centric development methodology defined in an international European project. By mapping the SOA process decisions against a core model of architectural knowledge [4], which is regarded as a common frame of reference for architectural knowledge sharing, this paper investigates to what extent this core model can be used to model process decisions.

This paper provides two original contributions. First, it highlights that process decisions are important architectural concerns by describing two examples of process decisions (with options and rationales) coming from the SOA domain and from industrial experience. Second, it concretely analyzes how process decisions can be codified by using architectural knowledge models aimed at representing decisions in general. Potential gaps are discussed, as a first step towards a more articulated analysis of process decisions (as part of architectural knowledge) modeling.

The rest of the paper is organized as follows. In Section 2 we provide an overview of the related work. To motivate our work, Section 3 discusses the SOA characteristics that cause additional SOA process decisions as opposed to traditional software engineering. We demonstrate the link between the SOA characteristics and process decisions in Section 4 by using two specific SOA process decisions. In Section 5, we map these two process decisions on the core model and ague that specific fields or notations are necessary for representing SOA process decisions. Our conclusions are given in Section 6.

2. RELATED WORK

Our work aims at investigating to what extent the current architectural knowledge models can be adapted to represent decisions on software development processes, especially in the service oriented context. Therefore, we position ourselves on the fields of architectural knowledge modeling and service oriented development.

The need to record the rationale behind design decisions has been underlined by Potts & Bruns already in the 80's [22]. Since then, we observe a major trend towards architectural design decisions that regard architectural design as a product. A template for modeling architectural design decisions proposed by Tyree [24] is one of the most well known meta models. This template captures the architectural design decisions by recording design issues, assumptions and constrains for the resulting system, arguments for making decisions, its implications and its relationship with other decisions and artifacts. This model consists of self-explaining entities and therefore is very easy to be used for representing architectural design decisions in general.

For complex, software-intensive systems, an ontology of architectural design decisions proposed by Kruchten [15] turns out to be useful. This model methodically classifies architectural design decisions into three kinds namely: existence decisions, property decisions and executive decisions. The properties, attributes and relationship between the decisions can be completely captured by this model.

Comparing these two models, many entities in the models address same concepts but are named differently. For instance, both of these two models address the reasoning on the justification of a decision. However, this reasoning is named as *argument* in Tyree's template but *rationale* in Kruchten's ontology. Therefore, a core model of architectural knowledge, which can be used as a common terminology for representing architectural knowledge, is presented in [4]. Each element in the core model can be mapped onto the concepts that are used in other existing models. As stated in [4], the current models can be "wrapped around the core model to specialize the core concept architectural design decision".

As explained in Section 3, process decisions that regard development process or methodology as a product are put much less emphasis in architectural knowledge modeling. Past research did consider process decisions by documenting the end-result of the decision process (in terms of e.g. process models or guidelines and best practices). Though, the reasoning behind such process decisions remained undocumented.

The service oriented paradigm becomes more popular and dominant in modern software engineering. Since service engineering has not reached a high level of maturity, both industry and academia are working on the development process for service oriented systems. Therefore, the role of process decisions that are generated in methodologies and life cycle models for service engineering becomes increasingly important. Process decisions actually have been used more or less implicitly for years. Best practices and guidelines [6] towards service oriented development are often addressed in the industry. Methodologies [20] approaches [14] and life cycle models [11, 3] for service oriented system development are also discussed in the literature. Although these authors mention and use process decisions within their work, they do not explicitly address process decisions from an architectural knowledge point of view, which leads a partial picture.

Zimmermann *et al.* [26] present some interesting work on SOA design decisions. They suggest to "position architectural decision modeling as a prescriptive service realization technique". SOA process decisions are recognized as one type of SOA design decisions. However, their decision tree is mainly intended to guide the practitioner through the design process rather than as a model for representing architectural knowledge.

3. MOTIVATION

Only when architectural knowledge is explicitly documented and stored, is it available for sharing among stakeholders. These stakeholders could be in the same development team, in the same organization or even beyond the organization. Knowledge is of great business value for a company [7].

Process decisions, one type of architectural decisions, are paid little attention to in architectural knowledge modeling in traditional software engineering. This is understandable because software life cycle models have reached a high level

of maturity. The development processes are well known and few process decisions have to be made. The significance of documenting these process decisions is therefore relatively low.

However, process decisions play a more important role in SOA. First of all, process decisions become a must. Once standard software development process models no longer fit, more process decisions have to be made for the development methodologies [2]. Various authors have devoted their effort to propose service life cycle models [11, 3] and agile design and development methodologies [20] for developing service centric applications. For instance, whether the process of service discovery should execute at design time or runtime is on the one hand an architectural design decision and on the other hand a process decision. When regarding the system as a product, we need to decide on whether the system will support dynamic service discovery. When regarding the process as a product, the decision is on process-related aspects. For instance, when services are discovered at design time, service engineers need to first discover available services and then decide whether to reuse the services or develop new services. In this case, the process of service discovery is carried out by a service engineer. When services are discovered at runtime instead, the process of service discovery is carried out automatically by a composition engine and under the control of a service provider. The process-related aspect in this example is the executor of the process (a service engineer or composition engine controlled by a service provider). Such types of decisions yield two dimensional aspects, the process aspect orthogonal or complementary to the architectural aspect.

Secondly, new characteristics of the service oriented paradigm enforce process decisions. Examples of these new characteristics include: software as services [19], shift of the ownership of software [8], coordination across and beyond the boundaries of an organization [18], harder requirements for runtime environment [21]. These characteristics require extra considerations when taking process decisions. For instance, as stated in [16], existing architectural views are incomplete because the owner of the system as one of the stakeholders is often not addressed. And systems often depend on other systems, with possibly different owners. If you then want to change one of them, you have to come to an agreement with the owners of other systems, since they have to incorporate the same change, at the same time. As a result, we are studying process decisions in the service oriented paradigm as a modern software engineering discipline.

Similar to design decisions in other fields, process decisions have to be documented. Here, the question we pose ourselves is: can the existing models for representing architectural knowledge in general be used to represent process decisions in the SOA context too?

To answer this question, we are in the process of eliciting a number of process decisions from an international project in the SOA domain. We analyze these decisions and document all the relevant information for each of them. Our hypothesis is that if the concepts that we have used in documenting those process decisions can be all mapped onto the existing architectural knowledge models, process decisions in the SOA domain can therefore be codified using these models. Suppose some concepts can not be mapped onto the existing models, we conclude that process decisions face new challenges in architectural knowledge modeling.

4. SAMPLE SOA PROCESS DECISIONS

In this paper, we particularly focus on process decisions that occur in service oriented development. To give some factual examples, we take the SeCSE European project [23] as a case study. In this project, the SeCSE methodology has been defined and has served as the main developing process for the SeCSE industrial and academic partners to build service centric systems.

As explained in Section 3, SOA process decisions are mainly caused by the new characteristics introduced by the service oriented paradigm. There we examine two characteristics, namely 1) shift of the ownership of software and 2) coordination across and beyond the boundaries of an organization. Each characteristic is illustrated by a process decision elicited from the deliverable of the SeCSE methodology.

4.1 Shift of ownership of software

The concept of software as a service (SaaS) in the service oriented paradigm leads to the shift of the ownership of software [8]. Software is no longer required to be pre-installed on the user's computer. Rather, software as a service is based on service level agreements (SLAs). Instead of buying (owning) the service, the user uses (rents) the service from a vendor. The user of the service is not the owner of the software anymore. Instead, the service provider (vendor) is the owner of the service. This shift of ownership not only changes the way software is delivered, but also the way software is maintained and upgraded.

Traditionally, users of software pay for maintenance contracts. When faults are detected or requirements changed, users have to ask for support from vendors. If software systems are developed in-house, the users of the systems are seeking support from the development team. This procedure is time consuming and costly. In the service oriented paradigm, both users and vendors benefit from the transfer of ownership. From the users' perspective, users are freed from taking the time and initiative for requesting support from the vendors. From the vendor's perspective, time and cost are saved because as soon as one service is improved, all the users of that service have the benefit of the improvement.

In traditional software engineering, maintenance is a standard process in the software life cycle model. In service engineering instead, more process decisions need to be made. For instance, a process decision must be made regarding how a service can be updated without creating conflicts among various service consumers who share the same service.

Moreover, in the traditional approach it is very difficult to switch to a different software system when users are not satisfied with the current anymore. Discarding current software is a huge waste of investment. In the service oriented paradigm, users are able to easily move to any other service as long as the functionality and quality requirements are met. Therefore, extra process decisions must be considered to ensure that the infrastructure of the system supports users to move to other services if they want.

The third difference introduced by the shift of the ownership of software lies in the control on software. Traditionally, a vendor was not allowed to modify or remove the software without the permission of the user. Instead, in the service oriented paradigm, vendors can modify or remove services without notifying the users. Despite the SLAs between service providers and service consumers, service consumers are often concerned about the ability of the system to recover

from services failure. This difference proposes process decisions to ensure that the infrastructure of the system supports system recovery.

To give an example of process decisions that need to be made due to the differences that are explained above, we elicit a process decision from the deliverable of the SeCSE methodology. The process decision, its alternatives and the rationale are presented in Table 1. In this process decision, the possibility for users to easily switch to other services and the recoverability of the system are considered as the main drivers for making the decision.

4.2 Cross-organizational cooperation

Cross-organizational cooperation is an industrial practice since several decades, accordingly companies decide to acquire non-core functions externally rather than develop the necessary technology internally [17, 10]. In this approach, tasks are delegated to specialized third parties [13]. The main drivers for doing so include cost efficiency and production reorganization [10]. Such trend is called IT outsourcing [17]. As a management approach, it has been widely applied to software development.

The cross-organizational cooperation in the service oriented paradigm is different. The stakeholders coexist in the SOA domain. Instead of an active-passive relationship (outsourcer and supplier) in the outsourcing approach, the stakeholders coexist in the SOA domain. Process decisions that are made by outsourcers have minor impact on suppliers and vice versa. For instance, a supplier choosing to use the waterfall life cycle model to develop the assigned piece of software does not impact the outsourcers.

We identify three types of cross-organization interaction in the service oriented paradigm. First, a service oriented system involves a number of stakeholders and these stakeholders act by assuming various roles at the same time. For instance, services are developed by a service engineer; hosted and published by a service provider; registered by a service broker, integrated into a system by a service integrator and invoked by service consumers. All these stakeholders must cooperate in such a way that the resulting system not only fulfills the end user's requirements but also meets the individual stakeholder's requirements. Second, one service can be integrated by various service integrators and invoked by various service consumers. Service providers have to ensure that changes that are made to the service under the request of one service consumer do not impact the usage of the other consumers. Third, services can be selected from various service providers. Therefore, a service consumer has to deal with a number of service providers for SLAs.

The cross-organizational cooperation poses new challenges in process decisions in the context of SOA. The main reason is that process decisions not only influence the development process, but also have impact on various stakeholders. When arguing about rationale for the decision, we have to keep in mind that other stakeholders might be involved.

To give an example of how process decisions in the context of SOA influence multiple stakeholders, a sample process decision has been elicited from the deliverable of the SeCSE methodology. In this example, a choice needs to be made on which stakeholder should perform service monitoring. As illustrated in Table 2, choosing a third party for service monitoring impacts the tasks that service provider and service consumer should carry out. The example therefore demonstrates how cross-organizational cooperation demands process decisions in the context of SOA.

5. MAPPING SOA PROCESS DECISIONS ON THE CORE MODEL

5.1 Gaps in the existing architectural knowledge models

Two sample process decisions in the context of SOA are presented in Section 4.1 and 4.2. Each process decision is explained by means of *concern*, *ranking criteria* and a number of *architectural decision alternatives*. Each alternative decision is described in terms of *identifier*, *description*, *executor of the decision*, *related stakeholders*, its *relationship(s)* to the other decisions, *status* and *rationale*.

An architectural decision must be made when there is a *concern* over the resulting system and optional solutions exist to overcome the concern to some extents. These optional solutions are named as *architectural decision alternatives* in the examples. To judge which alternative is the best for the system, *ranking criteria* usually reflect the demand of the user and therefore can be used as measurement for making decisions. For the sake of clear representation, each alternative decision has an *identifier*, which can be used as reference. Usually an identifier consists of a sequence number and a title of the decision. The *description* of the alternative decisions provides more explanation on the decision than the title. The *rationale* of a decision is of paramount importance for reasoning. Pros and cons of the decision are typically explained there. The state of the decisions, such as discarded, approved or under discussion, is recorded in *status*. If the decision is taken, its relationship(s) with the other decision(s) can be indicated in *relationships*. The stakeholder(s) who react(s) on the decision is called the *executor of the decision* and the stakeholder(s) who is (are) impacted by the decision is called *related stakeholders*.

From architectural knowledge modeling point of view, the concepts that we use in the tables form a conceptual model for recording process decisions. The two sample process decisions are instances of the conceptual model. In order to analyze whether the current architectural knowledge models can be used to codify process decisions as well, we map the concepts that we have used for representing the process decisions onto the entities that are used in the existing architectural knowledge models. The core model, the 'Esperanto' of architecture knowledge, can be used as a common frame of reference [4].

Therefore, we intend to map the concepts that are used in the sample process decisions onto the core model as our first step. In doing so, we are able to locate most of the concepts in the core model. For instance, "ranking" refers to *ranking criteria*; "alternative" in the core model refers to *architectural decision alternatives* including *identifier* and *description*; a decision with the *status* of approved is an "architectural design decision" in the core model; the decision loop in the core model captures the *relationship(s)* between the decisions, etc. The only concepts that we are not able to position in the core model are *executor of the decision* and *related stakeholders*.

It is not surprising to see that current architectural knowledge models do not explicitly address stakeholders who execute the decision and stakeholders who are impacted by the

Table 1: A SOA process decision due to the shift of ownership of software

Concern		The architecture of a system must support service discovery in different ways. When the service consumer fails to execute the services, the infrastructure of the system must ensure that the system can be recovered from the failure.
Ranking criteria		Flexibility, reliability, recoverability, performance
Architectural decision alternative	Identifier	#1.a Static service discovery
	Description	In static service discovery, services are looked up during design time and their location is hard coded in the software.
	Executor of the decision	Service engineer, service integrator
	Stakeholders who are related to this decision	Service developer: The services that are discovered in this process influence the work of a service developer. The more services that are selected, the less development work the service developer has to perform.
	Relationship(s)	
	Status	Discarded
	Rationale	Pros: - Static service discovery has better run-time performance since no communication between service consumers and the service registry has to occur at run-time. Cons: - The failure of a particular service may lead to the failure of the whole system. - Often service consumers are not aware of new services that have similar functionalities but better qualities in the service registry because the location of the discovered services is hard coded in the software. - Manual configuration is required when services are migrated.
Architectural decision alternative	Identifier	#1.b Dynamic service discovery
	Description	In dynamic service discovery, services are looked up dynamically at runtime by service search.
	Executor of the decision	Composition engine
	Stakeholders who are related to this decision	the Service Monitor (who measures and assesses the service QoS and initiates this process indirectly)
	Relationship(s)	
	Status	Discarded
	Rationale	Pros: - Alternative services that have similar functionalities but better qualities can be identified at runtime. - It is easy to switch other alternative services because their location is not hard coded in the software. Cons: - Communication between service consumers and the service registry at run-time impact the performance of the system. The failure of a particular service may lead to the failure of the whole system.
Architectural decision alternative	Identifier	#1.c Combination of static and dynamic service discovery
	Description	Adopt static service discovery in conjunction with dynamic service discovery. Perform static service discovery at design time while allowing dynamic service discovery for recovery at run-time when errors occur.
	Executor of the decision	Service engineer, service integrator, Composition engine
	Stakeholders who are related to this decision	the Service Monitor (who measures and assesses the service QoS and initiates runtime discovery indirectly)
	Relationship(s)	
	Status	Approved
	Rationale	Pros: - Since the services that are required in the system are discovered at design time, no extra communication is needed between service consumers and the service registry. Therefore, the performance remains high. Only when services those become unavailable or fail to meet certain requirements at the run-time, dynamic service discovery is triggered for discovery of alternatives. Although the run-time performance gets lower due to extra communication, the reliability of the system is enhanced.

Table 2: A SOA process decision due to the cross organization cooperation

Concern		Service monitoring is of great importance for a successful service oriented system because it observes and supervises at runtime the actual behavior of the services. The system must ensure that the monitoring data can be collected by both the service provider and the service consumer.
Ranking criteria		Complexity, effectiveness, trustworthiness, performance
Architectural decision alternative	Identifier	#2.a Service monitoring is performed by the service provider and the service consumer.
	Description	Both of the service consumer and the service provider perform service monitoring against the predefined QoS or SLAs. The former focuses on monitoring the consumption of the service in order to determine whether its behavior abides by the agreed QoS or SLA. The latter is more interested in the service execution.
	Executor of the decision	Service provider, service consumer
	Stakeholders who are related to this decision	
	Relationship(s)	#1.c
	Status	Approved
	Rationale	Pros: - Both of the service provider and the service consumer perform service monitoring according to their own concerns. For instance, the service consumer does not need to collect data about how the increasing number of service consumers affect the performance of the services. The separation of the concerns makes the monitoring task less complicated and more efficient. - When service monitoring data shows that a service stops response or performs incorrectly, service recovery procedure including runtime service discovery can be triggered immediately. Cons: Monitoring for own concerns lacks of trustworthiness. For instance, when the service consumer detect that a particular service does not fulfill the performance requirements as it is defined in the SLA, the service provider might not agree with the claim. Conflict may occur between the service provider and the service consumer.
Architectural decision alternative	Identifier	#2.b Service monitoring is performed by a delegated trusted third party that both the service provider and the service consumer agree with.
	Description	A delegated trusted third party is agreed by both the service consumer and the service provider to monitor the services. This third party regularly reports monitoring data to the service consumer and the service provider with respect to their concerns.
	Executor of the decision	A delegated trusted third party
	Stakeholders who are related to this decision	Service provider, Service consumer
	Relationship(s)	
	Status	Discarded
	Rationale	Pros: Both the service provider and the service consumer are freed from the task of service monitoring. The report from the third party can be used as an evidence for charging the service consumer or penalize the service provider based on the SLAs. Cons: - Both the service provider and the service consumer have to agree upon the delegated trusted third party. Extra effort is required for selection and setting up contracts. - Extra communication between the delegated trusted third party between the service provider and service consumer has to occur. Monitoring data has to be reported on a regular basis. Both of the service provider and the service consumer have to response (if necessary) after the receiving of the monitoring data. - Service monitoring is a prerequisite for runtime recovery. The involvement of a third party introduces complexity. The failure of the service monitoring and the loss of communication with the third party influence the success of the whole system. Both the service provider and the service consumer do not have control on any problem that might occur.

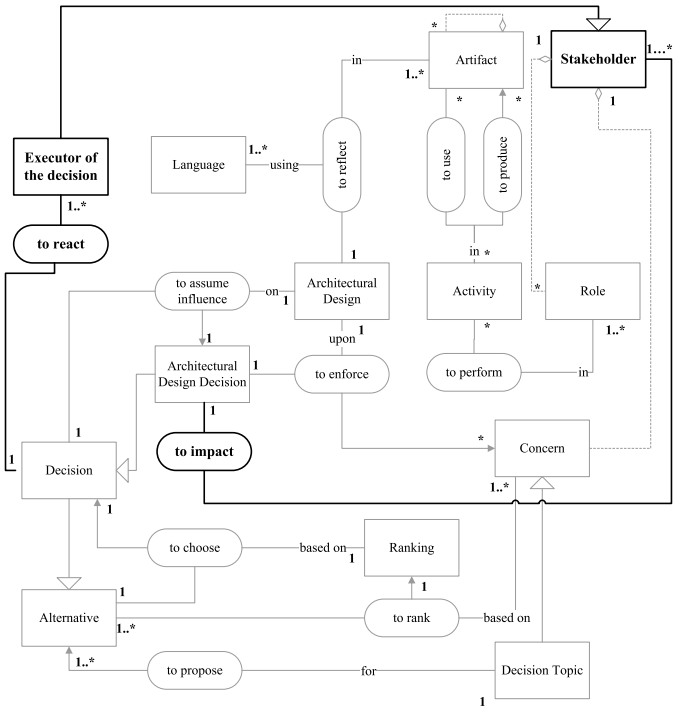

Figure 1: Extended core model for SOA process decisions

decisions. These models work well in traditional software engineering when aiming at representing decisions regarding architectural design as a product. These design decisions do not impact stakeholders but the architectural design only. For instance, choosing the client/server model is a design decision which will be executed by lower-level designers and implementors. This decision does not have direct impact on the stakeholders of the system.

However, when the ownership of software starts to shift to the service provider and the increasing cross-organizational cooperation happens in the SOA context, more stakeholders can exercise choices of architectural consequences. Decisions are not only made for the architectural design but also about the development process itself. As we can see in the sample process decisions in Tables 1 and 2, various stakeholders are involved in development process and they play more important role in making process decisions. Therefore, architectural knowledge models need to make these executors explicit. This shows a gap in the current architectural knowledge models for representing process decisions in the SOA context.

5.2 A suggestion to extend the core model

Process decisions introduce more concepts than the design decisions that are commonly addressed by the existing architectural knowledge models. To overcome this gap, we suggest to extend the these models. There we suggest a way to extend the core model for modeling architectural design decisions including process decisions in the SOA context. Figure 1 illustrates our suggested solution. Firstly, an entity named "executor of the decision" can be added in the core model. The entity "Stakeholder" is the generation of "executor of the decision". A link with the action "execute" between the added entity "executor of the decision" and "architectural design decision" can be created. This link points out that "executor of the decision" will react on the decision. Furthermore, a link with the action "impact" between the entity "architectural design decision" and "stakeholder" can be created to denote that architectural design decisions impact stakeholders. In this way, the core model can be extended to serve as a common language to model architectural knowledge including SOA process decisions.

The two missing entities that we identified are just a starting point. More characteristics are introduced by the service oriented paradigm eventually causing more entities to be found missing in the current architectural knowledge models. The suggested extension of the core model serves as an example showing how to overcome the inadequacy of modeling the elicited SOA process decisions that are discussed in Section 4. Further analysis on SOA characteristics and their related process decisions need to be done to gain a complete picture on process decisions in the SOA context.

6. CONCLUSIONS AND FUTURE WORK

Architectural knowledge must be modeled for sharing. Architectural design decisions, a key element in architectural knowledge, attract increasing attention from both the industry and academia. Much work has been done on modeling architectural design decisions and especially regarding architectural design as a product. However, when modern computing paradigms such as the service oriented paradigm emerge, process decisions as part of architectural design decisions introduce new challenges for architectural knowledge modeling.

The purpose of this paper is to: a) emphasize the necessity and importance of making process decisions in the SOA context and b) point out a gap in the existing architectural knowledge models for modeling SOA process decisions. To serve these purposes, two examples of SOA process decisions coming from industrial experience are discussed. How these two process decisions are related to the characteristics introduced by SOA is explained. We have analyzed how to use an architectural knowledge model to represent these two SOA process decisions. The two concepts we are missing in the analyzed model are both related to the stakeholders of a system. We recognize that this is in line with the importance of identifying stakeholders and concerns in most current architectural work. Finally, an extension of the core model is suggested as a first step toward the solution.

So far we used just one architectural knowledge model for our mapping. We also studied the architectural models proposed in [24, 15, 26]. Still, we cannot locate yet the two missing concepts in the other models in a straightforward manner. Further work needs to be done in this direction.

Although modeling process decisions by customizing current architectural knowledge models can be generalized in the other domains as well, we specifically focus on the SOA context. This paper discusses two characteristics introduced by SOA. When more characteristics will be studied, we might find more entities that challenge the current architectural knowledge models. In our work, we plan to conduct a number of industrial case studies to gain a more thorough picture on SOA design decisions.

Acknowledgments

We would like to thank Elisabetta Di Nitto and the SeCSE team for disclosing their deliverables and providing precious feedback.

7. REFERENCES

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, Reading, 2003.

[2] G. Benguria, A. Ana Belén, D. Sellier, and S. Tay. European cots user working group: Analysis of the common problems and current practices of the European cots users. In *COTS-Based Software Systems*, pages 44–53. Springer Verlag, 2002.

[3] M. B. Blake. Decomposing composition: Service-oriented software engineers. *IEEE Software*, 24(6):68–77, 2007.

[4] R. C. d. Boer, R. Farenhorst, P. Lago, H. v. Vliet, V. Clerc, and A. Jansen. Architectural knowledge: Getting to the core. In *the Quality of Software-Architectures (QoSA)*, Boston, USA, 2007.

[5] J. Bosch. Software architecture: The next step. In *Software Architecture*, volume 3047/2004, pages 194–199. Springer Berlin / Heidelberg, 2004.

[6] A. W. Brown, M. Delbaere, P. Eeles, S. Johnston, and R. Weaver. Realizing service-oriented solutions with the IBM rational software development platform. *IBM systems journal*, 44(4), 2005.

[7] G. R. Djavanshir and W. W. Agresti. It consulting: Communication skills are key. *IT Professional*, 9(1):46–46, 2007.

[8] A. Dubey and D. Wage. Delivering software as a service. *The McKinsey Quarterly*, 2007.

[9] R. Farenhorst. Tailoring knowledge sharing to the architecting process. *ACM SIGSOFT Software Engineering Notes*, 31(5):3, 2006.

[10] F. Franceschini, M. Galetto, A. Pignatelli, and M. Varetto. Outsourcing: guidelines for a structured approach. *Benchmarking: An International Journal*, 10(3):246–260, 2003.

[11] Q. Gu and P. Lago. A stakeholder-driven service life cycle model for SOA. In *2nd international workshop on Service oriented software engineering: in conjunction with the 6th ESEC/FSE joint meeting*, pages 1–7, Dubrovnik, Croatia, 2007. ACM.

[12] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, page 109, Washington, DC, USA, 2005. IEEE Computer Society.

[13] A. Kakabadse and N. Kakabadse. Outsourcing: Current and future trends. *Thunderbird International Business Review*, 47(2):183–204, 2005.

[14] Y. Kim and H. Yun. An approach to modeling service-oriented development process. In *IEEE International Conference on Services Computing (SCC'06)*, pages 273–276, 2006.

[15] P. Kruchten, P. Lago, H. v. Vliet, and T. Wolf. Building up and exploiting architectural knowledge. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, page 291, 2005.

[16] N. Lassing, D. Rijsenbrij, and H. v. Vliet. Viewpoints on modifiability. *International Journal of Software Engineering and Knowledge Engineering*, 11(4):453–478, 2001.

[17] J.-N. Lee, M. Q. Huynh, R. C.-W. Kwok, and S.-M. Pi. It outsourcing evolution: past, present, and future. *Communications of the ACM*, 46(5):84–89, 2003.

[18] H. Ludwig. Web services qos: External slas and internal policies or: How do we deliver what we promise? In *Web Information Systems Engineering Workshops*, volume 00, page 115, 2003.

[19] M. P. Papazoglou. Service -oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISEŠ03)*. IEEE Computer Society, 2003.

[20] M. P. Papazoglou and W.-J. v. d. Heuvel. Service-oriented design and development methodology. *Int. J. Web Engineering and Technology (IJWET)*, 2(4):412–442, 2006.

[21] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing research roadmap. *Dagstuhl Seminar Proceedings 05462 (SOC)*, 2006.

[22] C. Potts and G. Bruns. Recording the reasons for design decisions. In *International Conference on Software Engineering*, Proceedings of the 10th international conference on Software engineering, pages 418 – 427, Singapore, 1988. IEEE Computer Society Press.

[23] SeCSE. Secse website http://secse.eng.it/, 2007.

[24] J. Tyree and A. Akerman. Architecture decisions: Demystifying architecture. *IEEE Software*, 22(2):19–27, 2005.

[25] J. Ven, A. Jansen, J. Nijhuis, and J. Bosch. Design decisions: The bridge between rationale and architecture. In *Rationale Management in Software Engineering*, pages 329–348. Springer Berlin Heidelberg, 2006.

[26] O. Zimmermann, J. Koehler, and F. Leymann. The role of architectural decisions in model-driven SOA construction. In *the 21st Int. Conf. on Object-Oriented Programming, Systems, Languages, and Applications,*, Best Practices and Methodologies in Service-Oriented Architectures, Portland, 2006. ACM.