Modeling the Interactions between Decisions within Software Architecture Knowledge

Mohamed Soliman and Matthias Riebisch

Universität Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{soliman,riebisch}@informatik.uni-hamburg.de

Abstract. Software architecture is developed as a result of a selection process for software architectural solutions. The complexity, diversity and evolution nature of architectural solutions' interactions forces the architect to make critical design decisions based only on his own experience. Even though, the same design problem has already been addressed by another architect in a similar situation. In this paper, we are presenting a model for reusable software architectural knowledge to support the architect within the design process in understanding the relationship between the different architectural solutions, and how they impact the architectural design reasoning. In addition, the model acts as a base for organizational software architectural knowledge sharing. Our contribution in this paper is classifying and modeling the solutions and decisions' interactions, as well as how the design decision can be used as a reusable element for sharing the architectural experience.

Keywords: Software architecture, design decision, architecture knowledge, design reasoning.

1 Introduction

The software architect is responsible on taking the most important design decisions within the software design process. These architectural decisions [1] must be identified early in the project lifecycle due to their long-term impact on the system quality, and their tenacious behavior, which makes them quite expensive to change [2]. Even with their well-known impact, the architect is forced to take design decisions based solely on his personal experience, due to the enormous amount of possibilities for interacting architectural solutions[1], that must be selected in a limited project budget and schedule. Morever, the current state of the art approaches for architectural knowledge and solutions lack the required support for analyzing the interactions between solutions. Within this situation, the architect is restricted in discovering the right series of architectural solutions, and analyzing their impact on the system quality. This leads to sub-optimal decisions, which can significantly influence the system quality.

[1] In this paper, we use the term 'architectural solution' to refer to the different solutions that the architects use, such as patterns, tactics, technologies and products.

P. Avgeriou and U. Zdun (Eds.): ECSA 2014, LNCS 8627, pp. 33–40, 2014.

One of the main reasons that promotes this problem is the heterogeneous nature of architectural solutions, such that it is hard to set a common handling during design between the different solutions. In the past two decades, several classes of architectural solutions were captured separatly in the current state of the art (e.g. architectural styles [3], patterns [3,4], tactics [2], unit operations and different technologies and products). Each class is concerned with solving different types of design problems using different notions. In addition, each class is described originally in a different way, such that its arduous to combine two members from different classes together. Nevertheless, a combination of various members is required to develop the system software architecture. Such a diversity nature of software architectural solutions represents a challenge within the design process, because each solution has its unique impact on the subsequent design decisions as well as on the behavior of other solutions.

In addition to the above-mentioned problems, the interaction between the architectural solutions and decisions are constantly evolving, through new design ideas that emerge everyday from the mind of the designers. Thus, maintaining an evolvable reusable architectural knowledge[2] would support the organisation to share the design experience and solutions between the different software architects. This objective is derived by the notion of characterizing the architecture design process as a knowledge intensive process [5], such that losing this knowledge, recollecting and transferring it again is an expensive process. This idea of maintaining an architectural knowledge would improve the quality and productivity of the software architecture design process within the organization through learning, verification and improvement of existing solutions.

In this paper, we are proposing a model for a reusable software architectural knowledge to support the architect and the organisations in reasoning about software architecture design, as well as maintaining and sharing architectural knowledge. We concentrated our work on trying to understand and model the impact of selecting an architectural solution on the subsequent architectural decisions and the reasoning process, as well as providing the fundamental elements for sharing the architectural design decisions among different projects. This paper is organized as follows. First, related work to architectural knowledge and solutions are presented and discussed. Then, our research steps are explained, follwed by our result model which is explained with several examples. The paper ends with a discussion, future work, and some concluding words.

2 Related Work

In the patterns community, pattern languages are proposed (e.g. [6]). However, the relationships between patterns are modeled in a high level, without specifying clearly how the patterns interact with each other. Harrison et al. [7] modeled the relationships between architectural patterns and tactics. Their approach

[2] we use the term 'architectural knowledge' to refer to the reusable information that supports the architect within the design process.

is based on relating the solutions through their impact on the system components. Such modeling supports the architect to describe the solutions within the software architecture. However, with less guidance on how to take the design decisions. Since the paradigm shift of modeling the software architecture as a set of design decisions [1], several models and tools [8] have been proposed. The main target for the former suggested approaches is to document and share the design decisions of a specific software system for the sake of preventing the software architecture erosion phenomena. The recent work by Zimmerman et. al. [9] and its extension [10] distinguish between project specific design decision outcomes and its reusable part of design issues and solutions. Zimmermann et al. formally described the relationships between the design issues and the architectural solutions, as a way to support the architect in the decision making process. Neverthless, the distinct behaviors of the different types of solutions are not explicitly described. To the best of our knowledge, there are no more recent work which address the mentioned problem. Therefore, our approach in this paper is an extension to the model proposed by Zimmerman et al. to address the aforementioned points.

3 Research Method and Steps

To achieve our goal, we followed an inductive qualitative content analysis process [11]. First, we analyzed the influence of selecting an architectural solution on the design reasoning independent from other solutions. Then, we experimented with the relationships between the different solutions. In order to implement these steps, we selected samples from the architectural solutions. Two main criteria were considered in the selection process: A) Diversity: The chosen solutions belong to different classes, B) Popularity and success in the industry. The chosen architectural solutions were the Layer architectural style, the MVC architectural pattern, the architectural tactics by Bass et al. [2], basic unit operations and the GoF design patterns. We performed our analysis through the design realization steps, descriptions and examples provided in the solutions' sources, as well as case studies, which used the mentioned architectural solutions.

4 Reusable Software Architectural Knowledge Modeling

Fig. 1 shows our view for a high level contextual diagram for the reusable architectural knowledge. The diagram shows how an architectural knowledge is used within an organisation. First, the architect - influenced by the stakeholders' concerns and constrains - utilize the data and the reasoning logic within the architectural knowledge in order to support him taking the design decisions of the system. This process is followed by or intervened with capturing and documenting the system design decisions, which would act later as a source for enriching the architectural knowledge with new design solutions or logic in a separate harvesting process. Our main goal in this paper is to model the reusable architectural knowledge in relation to other contextual entities.

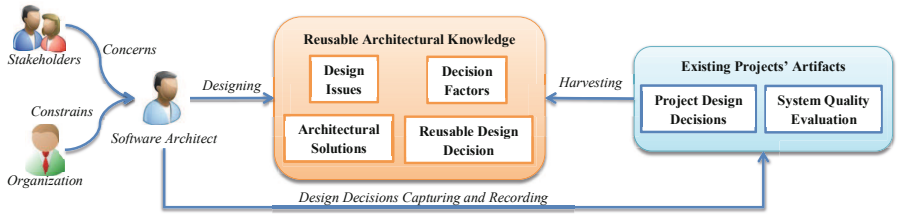

Fig. 1. Reusable Architectural Knowledge Context Diagram

We describe our model into two sections. The first section concentrates on the interaction between the architectural solutions and their influence on the design reasoning process, while the second section shows how the design decisions can act as a reusable component, in connection with existing software systems.

4.1 Solutions' Interactions within a Reusable Architectural Knowledge

Fig. 2 shows the proposed model. In the core of this model is the *Design Issue* concept, which represents the architectural design problems [3] that the architects need to solve, and associated to each design issue, there is a set of *alternative architectural solutions* which address this design problem [9]. Each architectural solution has a different impact on the quality attribute of the system, as well as a different impact on the resulting structure of the system components. Based on our described analysis process, we classified the architectural solutions into two main types:

1. **Triggering Architectural Solutions**: They are the type of solutions that have the ability to trigger new architectural design issues, such that in order to complete the architectural design of these solutions, new architectural design issues must be addressed. In this group belong architectural styles and architectural patterns.
2. **Elementary Architectural Solutions**: These are architectural solutions that do not trigger new architectural design issues. They are either as architectural unit operations (e.g. Component Decomposition) or solutions recommendations for a subsequent detailed design (e.g. Design Patterns [4]).

Architectural tactics [2] have a different nature as other solutions, such that it is hard to classify them all in a single group. Therefore, we divided the tactics among the two groups, into elementary and triggering tactics. Elementary tactics are tactics that do not triger new architectural design issues. For example, to improve the performance of a well-known process (e.g. Products sorting), selecting or changing the algorithm usually would not produce new architectural

[3] We diffrentiate between an architectural design issue and other detailed design or implementation issues, based on the software architecture definition of Bass [2].

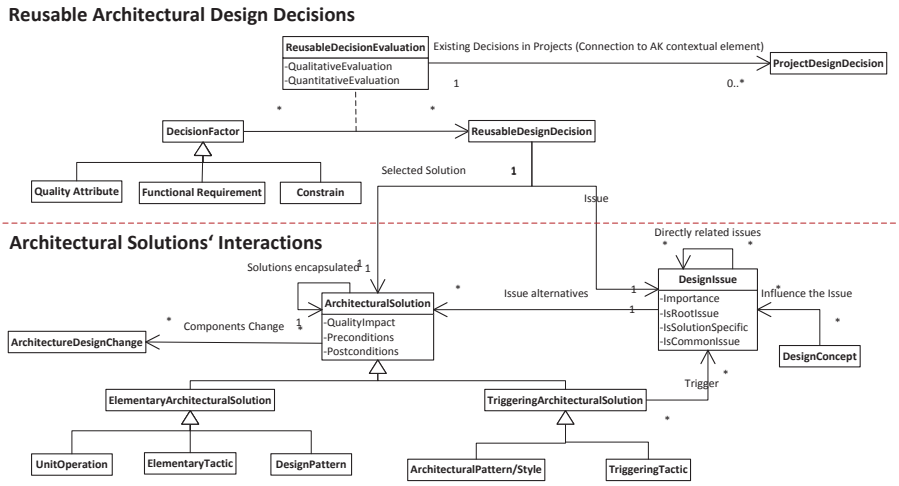

Fig. 2. Reusable Architectural Knowledge Domain Model

design issues, however, it can produce algorithmic or implementation issues. On the other hand, triggering tactics require more architectural design issues to be addressed in order to realize the design of the tactic. For example, improving the performance through caching, this would require to answer other design questions such as: which and where to cach the data? and how to synchronize the cached data?

Architectural design issues vary in their importance, types, scope and position within the reasoning process. Zimmerman et. al classified design issues based on their abstraction level. In order to support the architect in understanding when design issues occur within the reasoning process. We propose a classification for design issues, based on their occurance within the design reasoning process and their relationship to the architectural solutions.

1. **Root Design Issues**: They are design issues which are stimulated independently from previously selected architectural solutions. Enterprise or principal high level design issues (e.g. deciding the high level architectural style of the system or the main implementation technology) are popular examples that belong to this group.
2. **Solutions-Triggered Design Issues**: They are design issues that must be triggered based on a stimulation from a previously selected architectural solution. We further classified these issues based on their relationship to the architectural solutions into the following groups:
   (a) **Solution-Specific Design Issues**: They can only be triggered as a result of selecting a specific solution. They can't be triggered by any other solutions.
   (b) **Joined Design Issues**: A common design issue which can be triggered by different solutions.

(c) **Integration Design Issues**: This is a type of design issue which is conditionally triggered as a result of selecting two or more architectural solutions. It represents the integration design problem between the different architectural solutions.

Fig. 3 shows an example of a subset of issues and solutions that are triggered as a result of selecting the Layer architectural style and the MVC architectural pattern. Both solutions were triggered as a result of two root design issues, independently from any previous solution selected. However, they are influenced by several decision factors (e.g. requirements, team structure, ...). In order to realize the design of both triggering solutions, several design issues have to be addressed. For example, to define the Layer structure, an abstraction paradigm (e.g. distance from hardware or complexity) must be defined, this decision can depend on several factors (e.g. system domain). Similarly, in order to design the relationship between the Model and Views/Controllers within the MVC pattern, a 'change propagation mechanism' (e.g. using a Publish-Subscribe pattern) must be selected. Both of these issues are examples of solution specific design issues. On the other hand, designing the domain components of the system is required to be addressed for both solutions, however, for two different purposes. Firstly, to define how objects are communicated between layers, and secondly to provide a separation of concerns between the Model and View components. Finally, the introduction of both the Layer and MVC solutions together triggers an issue, whose purpose is the integration of both solutions components.

4.2 Reusable Architectural Design Decisions

In contradiction to other models, which consider an architectural design decision only as a project-specific entity, we argue that design decisions taken within different projects constitute a part of a reusable software architecture knowledge, such that a design decision consists of three main elements; A) The design issue addressed by this decision, B) The selected architectural solution, and C) The decison factors which influence the selection of this architectural solution to the design issue. The combination of the three elements acts as a reusable tuple which can be used among other projects.

The quality and success of design decisions varies from one project to another, such that a design decision concerned with a certain design issue and influenced by the same factors may be supported by different architectural solutions with different qualities. Therefore, a quality measurement factor should be associated with each of the reusable architectural design decisions. This quality measurement factor is originally obtained from the actual system quality or an evaluation for the architecture of the harvested projects. Fig. 2 shows the connection of the three tuples that constitutes a reusable architectural decision within the proposed model, as well as how the quality evaluation values are associated to the decisions and related to it's original sources in referenced projects.

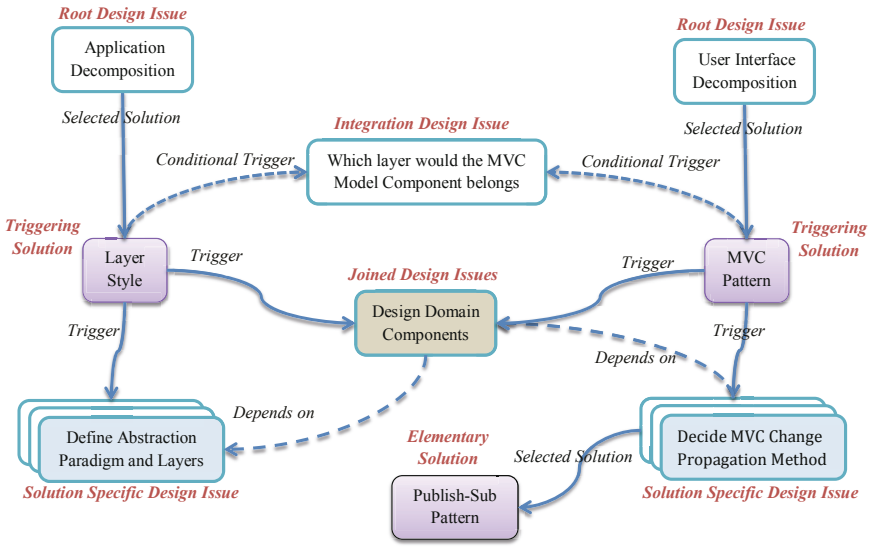

Fig. 3. An example showing the interaction of objects based on the proposed architectural knowledge domain model

5 Discussion and Future Work

In our analysis work, we selected solutions, which are well known and more used within the software development industry, specificly within the information systems domain. In addition, we considered various architectural solutions from different groups, in order to discover the different relationships and their impact on the design process. However, based just on this research, our results are not yet generalized to all types of architectural solutions in different domains. Throughout our analysis of solutions, we assumed that the design issues mentioned in their sources represent the expected solutions decisions. Neverthless, it is possible that other design issues can be generated in different other contexts, providing more relations.

The proposed model is a first step in our research plan, which seeks promoting ideas and solutions for the purpose of architectural knowledge sharing within organizations and architectural communities. Our plan involves several research steps: A) Developing a process to support the architect in utilizing the model, and how it relates with the existing design processes. B) Propose an approach for the harvesting process, to show how the project specific decisions can contribute to the architecture knowledge. C) Providing a tool support for the architect in using the model within the design process. We are working to verify our model in an industrial environment, through experimenting the model to support software architects in driving architectural designs.

6 Conclusion

The current state of the art describes the architectural solutions based on their impact on the system components, without addressing their influence on the decision making process. On the other hand, studies which are concerned with modeling architectural design decisions tend to describe decisions as project specific elements. We argue that using the design decision as a reusable element for modeling the interaction between the solutions, as well as for sharing the architecture knowledge, would support the architect within the design process.

In this paper, we made our first step towards our research objective. Through proposing a conceptual model for a reusable architectural knowledge. The model explained our classification of architectural solutions and design issues, as well as their interrelationships. In addition, it showed how an architectural design decision can be part of the reusable architectural knowledge, and how it would interact with other entities in its context. The model can assist the architects in selecting the right software architectural solutions path, and therefore, improving the quality of the produced software architecture.

References

1. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: 5th Working Conf. on Software Architecture, pp. 109–120 (2005)
2. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
3. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, 1st edn. John Wiley & Sons (July 1996)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.M.: Design Patterns: Elements of Reusable Object-Oriented Softwaresystemen. Addison-Wesley Professional (1994)
5. Lago, P., Avgeriou, P., Capilla, R., Kruchten, P.: Wishes and boundaries for a software architecture knowledge community. In: WICSA. IEEE Computer Society, 271–274 (2008)
6. Avgeriou, P., Zdun, U.: Architectural patterns revisited - a pattern language. In: Longshaw, A., Zdun, U., eds.: EuroPLoP, UVK - Universitaetsverlag Konstanz, pp. 431–470 (2005)
7. Harrison, N.B., Avgeriou, P.: How do architecture patterns and tactics interact? a model and annotation. Journal of Systems and Software 83(10), 1735–1758 (2010)
8. Shahin, M., Liang, P., Khayyambashi, M.R.: Architectural design decision: Existing models and tools. In: WICSA/ECSA. IEEE, pp. 293–296 (2009)
9. Zimmermann, O., Koehler, J., Leymann, F., Polley, R., Schuster, N.: Managing architectural decision models with dependency relations, integrity constraints, and production rules. Journal of Systems and Software 82(8), 1249–1267 (2009)
10. Capilla, R., Zimmermann, O., Zdun, U., Avgeriou, P., Küster, J.M.: An enhanced architectural knowledge metamodel linking architectural design decisions to other artifacts in the software engineering lifecycle. In: Crnkovic, I., Gruhn, V., Book, M. (eds.) ECSA 2011. LNCS, vol. 6903, pp. 303–318. Springer, Heidelberg (2011)
11. Elo, S., Kyngas, H.: The qualitative content analysis process. Journal of Advanced Nursing 62(1)(2), 107–115 (2007)