A Survey of Architectural Design Decision Models and Tools

Mojtaba Shahin[1], Peng Liang[2], Mohammad Reza Khayyambashi[3]

June, 2009
SBU-RUG-2009-SL-01

[1]Department of Computer Engineering and IT, Sheikh Bahaei University, Iran
[2]Department of Mathematics and Computing Science, University of Groningen, The Netherlands
[3]Faculty of Engineering, Department of Computer Engineering, University of Isfahan, Iran

Abstract

In the field of software architecture, there has been a paradigm shift from describing the outcome of architecting process mostly described by component and connector (know-what) to documenting architectural design decisions and their rationale (know-how) which leads to the production of an architecture. This paradigm shift results in emergence of various models and related tools for capturing, managing and sharing architectural design decisions and their rationale explicitly. This report intends to make a survey about the well-known existing architectural design decision models and their related tools.

Keywords: software architecture, architectural design decision, architectural knowledge

Content

1 Introduction

Software architecture has an important role to mange complicated interactions between stakeholders of software-intensive systems in order to balance all kinds of constraints [1]. The architecting process can be considered as a decision making process through which the appropriate decisions must be made at the right time [18]. Current methods for the documentation of software architecture concentrate on components and connectors [1], which causes problems such as expensive system evolution, and limited reusability of architecture due to the lost of design rationale knowledge [5]. Software architecture as a set of architectural design decisions (ADD) was proposed to address these issues [6]. The architectural design decisions, assumptions, and architectural design are integratedly making a concept called architectural knowledge (AK) which has been one of the controversial issues in the software architecture community recently [2,17]. Nevertheless, capturing and managing ADD through a systematic method can definitely improve the architectural capability of organization and also promote the interaction between stakeholders [36].

Practitioners and researchers have made great efforts to develop the models and related tools to capture, manage and share ADD explicitly [3,4,6,7,11,13,22,31,35]. Each of the models has its own strong and weak points. These models are similar to each other in several aspects and sometime use different terms for describing the identical concept. This situation leads to the problem of terminological misunderstanding across organizations or among different parts within an organization which might use different models to document their ADDs [13].

In this report we introduce and investigate nine well-known ADD models and related tools in order to express their capabilities and weaknesses.

In the reminder of this report, we first introduce the current ADD models in section 2, and in section, we investigate existing tools that support the ADD models analyzed in section 2. We conclude our work with future work directions in section 5.

2 Existing Models on Architectural Design Decision

The necessity for capturing rationale behind the design decisions was brought about by Pott and Burns in the late 1980s [10]. Since then, the tendency towards the research and application on architectural design decision and rationale, as a design product, increased gradually and many models and supporting tools are proposed and implemented [17].

2.1 Tyree Decision Template

One of the initial models in the field of ADD was proposed by Tyree and Akerman [3] which was used to model ADD as a text template. This template is to record the ADD by capturing *design issue*, *assumptions* and *constraints* of resulting system, *arguments* for making decisions, its *implications* and its *relationships* with other decisions and artifacts. This model has the self-explaining entities, e.g. *design issue*, *assumption*, and *constraints* etc., so it can be employed decently to represent the ADDs in general context, such as the ADDs in service oriented architecture (SOA) or in software product line.

2.2 Kruchten's Ontology

Kruchten introduced an ontology of architectural design decisions for complex and software-intensive system [7]. Every design decision can be placed in one of the three categories such as *existence decision*, *property decision* and *executive decision*. The design decisions have some common characteristic such as *rationale*, *scope*, *state*, *category*, *author*, *time-stamp* and *history* in this ontology. Moreover, every design decision might have dependencies, such as *constraints*, *forbids*, *enable*, *conflicts*, *is bound to* and so on with other decisions. Not only the design decisions are interrelated, but they are also related with other artifacts such as requirements and a part of implemented system. The dependencies can be *‘trace from/to’* and ‘*does not comply with’* [2], which carries simple semantic information.

2.3 Core Model

The models discussed in the sections 2.1 and 2.2 use different terms to describe similar and identical concept. For instance both of them describe the reasoning element behind the design decisions. It is called *Argument* and *Rationale* in Tyree’s template and Kruchten’s Ontology respectively. Using different terms to express identical concept hinder the effective management of ADDs due to the problem of terminological misunderstanding [13]. Thus, sharing and managing ADDs becomes a difficult task. The authors in [13] suggested a core model in order to overcome such issues. The core model has two basic characteristics: minimalistic and completeness. The first character refers that it should be impossible to express some concepts from the model in any other concepts from the model. For instance, the concepts of *Force* and *Architectural Driver* were in the previsions version of the core model, which are originated from *Concern* concept, have been removed [14]. The completeness character states that there are no concepts from other ADD approaches that have no counterpart in the core model. If there turns out to be such a missing concept, core model should be extended. For instance, when core model is applied in domain- or organization-specific context, it should be extended. A typical example is that, when applying core model in specific domain such as SOA, the core model faces new challenges for modeling ADDs [16].

2.4 Pattern-based Model

Although many models and tools have been introduced to record ADDs up to now, architects still have difficulties to record design decisions because of the need for substantial effort to document and maintain their decisions and even sometimes the architects do not know how to document their decisions [4]. Neil et al. have claimed that they can reduce the effort made to record design decision by using patterns. They compared pattern with Tyree’s decision template and noticed that a lot of characteristics of patterns match the entities of Tyree’s decision template. But architectural patterns can not alleviate software architect from all responsibility from documenting the ADDs. For instance, the architect should make the documentation personally for application-specific decisions.

2.5 Service-Oriented Architecture Decision Model

Zimmerman et al. work on the topic of using ADD on SOA [9]. They suggested to “*position architectural decision modeling as a prescriptive service realization technique*”. They employ design decisions models as a modeling complement for multi-purpose methodologies like Rational Unified Process (RUP) or agile methods like eXtreme Programming (XP) [12]. The authors claim that one of the reasons that current ADD models are not appealing to developers and architects, is the fact that the process of recording ADDs is considered as retrospective and unwelcome documentation tasks. These tasks do not provide any benefit during the original design work. In order to overcome this problem, the authors proposed the SOAD model [11], in which the *Architectural Decision* (*AD*) is the central entity of this model that expresses a single and concrete design issue. This entity has characteristics like: *name*, *scope*, *phase*, *decision drivers*, *problem statement*, *status* and *dependency relationship*. In this model every design decision has one or several *AD alternative* entities with specification of *pros* and *cons*. In this model the platform-independent *decisions* are separated from platform-specific ones for the implementation of SOA.

2.6 Archium Model

Archium [6] is a model that was introduced by Jansen et al. Architecture in Archium is described as and constituted by a set of decisions, (a *software architecture* = $dd_1 + dd_2 + dd_3 + \ldots + dd_n$, where dd_x is a design decision). This model consists of three sub-models: an architectural model, a design decision model and a composition model. The architectural model describes the software architecture, which is correspondent with components and connectors. The design decision model contains design decisions as first class entities. The composition model introduces required concepts to unite the two previous sub-models. The heart of the design decision model is the *Problem* concept, which describes the architecture problem together with *Motivation* and *Cause* concepts. The *Problem* is the goal that the ADD wants to solve. *Solution* contains the solutions that have been proposed to solve the problem. For each of the proposed solutions, *Description*, *Design rules*, *Design constraints*, *Consequences*, *Pros* and *Cons* are stored.

2.7 Architecture Design Decision Support System Model

Capilla et al. have suggested a model for architecting and evolving ADDs [22]. This model is based on the previous model in their prior work [21]. This model comprises three main parts: *Project Model*, *Architecture Model* and *Decision Model*. *Project Model* comprises the information related to the software architecture project described by one or more architectural views. The project decisions are explicitly documented as part of the "decision view". *Architecture Model* depicts the software architecture that is mostly described as a set of components and connectors. *Decision Model* is the core of the ADDSS model. In *Decision model*, attributes of decision are categorized as *Mandatory* and *Optional*. The mandatory attributes are those ADD attributes that are necessary to capture during the whole life cycle of system. The composition of these attributes constitutes the *Rationale* behind the design decisions. Some of these attributes are *Decision name* and *Description*, *Constraints*, *Dependencies*, *Status* and *Rationale*. It is up to particular organizations and architects to decide which of optional attributes could be more relevant to be stored as a part of ADD.

2.8 AREL Model

AREL (Architecture Rationale and Element Linkage) is a rationale-based architecture design model focuses on ADD reasoning process [31]. In AREL model, there are three key elements: *Design Concern*, *Design Decision* and *Design Outcome*. The inputs that cause or motivate a *design decision* are *design concerns.* Anything that influences the *design decision* can be a *design concern*, such as *non-functional requirement*. A *design decision* captures *design issues* and *design rationale*. AREL uses *qualitative* and *quantitative design rationale* to capture justification of *design decision*. *Design outcomes* are the results of a *design decision*. A chosen *design outcome* can become a *design concern* because it can create new design problems. AREL focuses on linking the problem space (*design concerns*) to the solution space (*design outcomes*) through *design decisions* in a uniform way.

2.9 DAMSAK

Babar et al. proposed a Data Model for Software Architecture Knowledge (DAMSAK) [35]. This model identifies and defines the architectural rationale constructs and their relationships, in order to support architecting process activities (e.g. architecture evaluation). This data model consists of twelve ADD elements, including *Architecture Decision*, *Architectural Significant Requirement (ASR)*, *Rationale*, etc. DAMSAK directly relate *architectural decisions* with *architectural scenarios* and *ASRs*, which can be used in architectural evaluation, for example in ATAM. DAMSAK can also support quantitative analysis, which uses the *Analysis Model* to systematically reason about the effect of different design *tactics* on *architectural scenarios*.

3 Tools Supporting the Existing ADD Models

In section 2, we have analyzed the existing ADD models and their structures were investigated. In this section we introduce the tools, which have been implemented based on various ADD models. Meanwhile, not all of the ADD models are supported by tools. For example, the creators of the Tyree's template have not offered any tool to support their model, and they use a text-based template for capturing design decisions.

3.1 Tool Supporting Kruchten's Ontology

Lee and Kruchten implemented a tool to support the Kruchten's Ontology introduced in section 2.2. They asserted that "*unlike many other ADD tools which acquire, list, and perform queries on decisions, our tool provides visualization components to help with decision exploration and analysis*" [19]. The tool has four main features: (1) The decision and dependency lists. It can list a set of current design decisions and their dependencies. Users can create, view, modify and delete the design decisions and their dependencies. (2) The decision structure visualization view, in this view the design decisions and their dependencies are visualized as a directed graph. Every decision is shown as a node, and the dependency from one decision to another decision is shown as a directed edge. (3) The decision chronology view, this tool also supports a time-base view for showing the design decisions. The goal of this view is to increase the understanding of the architecture's nature. By this view, a user can see the evolution of design decisions during a specified time interval. (4) The decision impact view, the goal of this view is to increase the understanding of architecture's dependencies on its set of design decisions. This view is valuable when radical changes are about to be made to a system, and makes the impact of certain changes obvious [20].

3.2 Knowledge Architect Tool Suite

The Knowledge Architect (KA) is a tool suite for capturing, sharing, translating and managing architectural knowledge [29]. This tool is one of the outcomes of the GRIFFIN project and supports the core model which was discussed in section 2.3. The KA tool suite entails the specialized support for integrating the various process activities and supports for collaboration between stakeholders [30]. At present, the KA tool suite comprises of 6 tools: Knowledge Repository, Document Knowledge Client, Excel Plug-in, Python Plug-in, Knowledge Explorer and Knowledge Translator. The most important capabilities that are provided by KA tool are: (1) capture (annotate), edit and view AK entities and their relationships; (2) search and retrieve semantically the AK entities and their relationships in Knowledge Repository; (3) translate the AK in various AK domain models from one to other and vice versa via Knowledge Translator; (4) share the AK entities with other users and stakeholders.

3.3 Architectural Decision Knowledge Wiki

AD_{kwik} (Architectural Decision Knowledge Wiki) is a model-based collaboration system that implements the SODA model introduced in section 2.5 [23]. AD_{kwik} tool is available on IBM alphaWorks [25] and similar to other wikis, the users only need web browser to work with the system. AD_{kwik} supports about 50 use cases. Some distinguished use cases are: (1) offer decision-making support by reusing appropriate decisions in the architectural decision repository; (2) import and export of decision content; (3) search and filter design decisions by *role*, *phase* and *scope* attributes; (4) decision lifecycle management for effective decision-making; (5) support collaboration features such as comments, tags, and attachments.

3.4 Archium

Archium is a design decision tool whose aim is to establish traceability between the architectural decision and other artifacts concepts, such as *requirements*, *decisions* and *implementations* [15]. All of these concepts are expressed by Archium language. Traceability helps everyone to get a better understanding from the architectural design. In this tool, the design decision is regarded as a “change function”. Archium can create the traceability between the design decisions and entities of architecture (components and connectors) easily, and keep it updated during life cycle of system. Dependencies between design decisions is shown by graph, however the type of dependencies is not expressed explicitly and all of the attributes of decision introduced in section 2.6 are listed and stored in a table of attributes.

3.5 Architectural Design Decisions Support System

ADDSS tool is a web-base tool to capture, maintain and document the architectural design decisions made during the architecting process [26]. This tool establishes the traceability between requirements and architectures via the decisions [26,27]. In this tool we can define one or many architectures for each project. Since the architectures are made as a result of iterative processes, users can store the design decisions of each iteration and also this tool allows the users to reuse well-known design patterns and styles placed in the tool, which is special feature provided by this tool.

3.6 AREL

AREL is a UML-based tool that aims in creating and documenting architectural design with a focus on architectural decisions and design rationale [32]. The most important capabilities provided by AREL tool are: (1) trace ADD from problem to solution space: users can trace *design outcomes* back to *design decisions,* and from *design decisions* back to *design drivers,* using the UML dependency relationship. (2) identify AK change impacts: user can identify all the ADDs and other AK elements, that are directly or indirectly implemented when AK is modified, based on the AREL causal model. (3) detect architectural design conflict: user can detect the design conflicts by looking at the missing links between *design concern* and *design outcomes* using the AREL causal model.

3.7 PAKME

PAKME (Process-based Architecture Knowledge Management Environment) is a web-based tool that supports the DAMSAK model introduced in section 2.9. The main services that are provided by this tool are: (1) knowledge acquisition service, this service provides forms to enter new generic (general scenarios, generic design decisions, architectural and design patterns) and project-specific (scenarios, design options, design rationale, and analysis findings) knowledge into the repository. (2) knowledge maintenance service, this service provides functions to update and instantiate the artifacts stored in the knowledge repository. (3) knowledge retrieval service, this service helps architecture users to find relevant information. (4) knowledge presentation service, this service provides architecture knowledge with templates and representation mechanisms (i.e. utility tree).

4 Conclusions and Future Work

The attention of software architecture community has changed over recent years and results in an increasing interest in ADD as one of key elements of architecting process. The aforementioned change brings the models, ontologies and related tools to capture, store, manage and share the AK, especially the ADD.

In this report we analyzed and compared the existing ADD models and related tools. First the models were compared with each other and their similarities and dissimilarities were manifested. Second the tools that support these models were compared with each other. The main results of this study are as follows:

- All of the ADD models treat the architectural design as a decision making process;
- Not all of the ADD models are supported by tools, some of them only use text-based template for capturing ADDs.

Our ongoing and future work focuses on several aspects: (1) the application and usage comparison of the ADD tools in practice and industrial environment (user satisfaction, cost and benefit, etc.); (2) the proposition of the general ADD framework that will have selected key features of existing ADD models; and (3) the guidelines on how to use the general ADD framework and related tools in the architecting process.

5 References

[1] Bass, L., Clements, P., and Kazman, R., Software Architecture in Practice, 2nd edition, SEI Series in Software Engineering, Addison-Wesley Pearson Education, 2003.

[2] Kruchten, P., Lago, P., and van Vliet, H., Building up and Reasoning about Architectural Knowledge. Proceedings of the 2nd International Conference on the Quality of Software Architectures (QoSA), pages 39-47, 2006.

[3] Tyree, J., and Akerman, A., Architecture Decisions: Demystifying Architecture, IEEE Software, 22(2):19–27, 2005.

[4] Harrison, N.B., Avgeriou, P., and Zdun, U., Using Patterns to Capture Architectural Decisions, IEEE Software, 24(4):38-45, 2007.

[5] van der Ven, J.S., Jansen, A., Nijhuis, J., and Bosch, J., Design decisions: The Bridge between Rationale and Architecture, In Rationale Management in Software Engineering, A.H. Dutoit, et al., eds, Springer, pages 329-346, 2006.

[6] Jansen, A., and Bosch, J., Software Architecture as a Set of Architectural Design Decisions, Proceedings of 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 109–120, 2005.

[7] Kruchten, P., An Ontology of Architectural Design Decisions in Software-Intensive Systems, Proceedings 2nd Groningen Workshop on Software Variability Management, Groningen, pages 109-119, 2004.

[8] Farenhorst, R., and de Boer, R.C., Core Concepts of an Ontology of Architectural Design Decisions, Technical Report IR-IMSE-002, Vrije Universiteit Amsterdam, 2006.

[9] Zimmermann, O., Koehler , J., and Leymann , F., The Role of Architectural Decisions in Model-Driven SOA Construction, Proceedings of International Conference on Object-Oriented Programming, Systems, Languages, and Applications, Best Practices and Methodologies in Service-Oriented Architectures, Portland, ACM, 2006.

[10] Potts, C. and Bruns, G., Recording the Reasons for Design Decisions, Proceedings of the 10th International Conference on Software Engineering (ICSE), pages 418–427, 1988.

[11] Zimmermann, O., Gschwind, T., Küster, J., Leymann, F., and Schuster, N., Reusable Architectural Decision Models for Enterprise Application Development. Proceedings of the 3rd International Conference on Quality of Software-Architectures: Models and Architectures (QoSA), pages 157-166, 2007

[12] Zimmermann, O., Koehler J., and Leymann F., Architectural Decision Models as Micro-Methodology for Service-Oriented Analysis and Design. Proceedings of the Workshop on Software Engineering Methods for Service-oriented Architecture (SEMSOA), CEUR-WS.org/Vol-244, 2007.

[13] de Boer, R.C., Farenhorst, R., Lago, P., Vliet , H. v., Clerc, V., and Jansen, A., Architectural knowledge: Getting to the Core. In the Quality of Software-Architectures (QoSA), Boston, USA, pages 197-214, 2007.

[14] de Boer, R.C., Farenhorst, R., van der Ven, J.S., Clerc, V., Deckers, R., Lago, P., and van Vliet, H., Structuring Software Architecture Project Memories. Proceedings of the 8th International Workshop on Learning Software Organizations (LSO), pages 39–47, 2006.

[15] Jansen ,A., van der Ven, J.S., Avgeriou, P., and Hammer, D.K., Tool Support for Architectural Decisions. Proceedings of the 6th IEEE/IFIP Working Conference on Software Architecture (WICSA), page 44-53, 2007.

[16] Gu, Q., and Lago, P., SOA Process Decisions: New Challenges in Architectural Knowledge Modeling, Proceedings of the 3rd international workshop on Sharing and reusing Architectural Knowledge (SHARK), pages 3-10, 2008.

[17] de Boer, R.C., and Farenhorst, R., In Search of 'Architectural Knowledge', Proceedings of the 3rd international workshop on Sharing and reusing Architectural Knowledge (SHARK), pages 71-78, 2008.

[18] Farenhorst, R., Lago, P., and van Vliet, H., Effective Tool Support for Architectural Knowledge Sharing, Proceedings of First European Conference on Software Architecture (ECSA), pages 123-138, 2007.

[19] Lee, L., and Kruchten, P., A Tool to Visualize Architectural Design Decisions, Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures (QoSA), pages 359-362, 2008

[20] Lee, L., and Kruchten, P., Visualizing Software Architectural Design Decisions. Proceedings of the 2nd European conference on Software Architecture (ECSA), pages 43-54, 2008

[21] Capilla, R., Nava, F., Pérez, S., and Dueñas , J.C., A Web-based Tool for Managing Architectural Design Decisions. ACM SIGSOFT Software Engineering Notes, 31(5):4-11, 2006.

[22] Capilla, R., Naval, F., and Dueñas, J.C., Modeling and Documenting the Evolution of Architectural Design Decisions, Proceedings of the 2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK/ADI), 2007.

[23] Schuster, N., AD_{kwik} – a Collaborative System for Architectural Decision Modeling and Decision Process Support based on Web 2.0 Technologies, Master thesis, Hochschule der Medien, 2008.

[24] Zimmermann, O., Schuster, N. and Eeles, P., Modeling and Sharing Architectural Decisions, Part 1: Concepts. IBM DeveloperWorks, August 2008.

[25] Schuster N., and Zimmermann O., Architectural Decision Knowledge Wiki, IBM AlphaWorks, March 2008, http://www.alphaworks.ibm.com/tech/adkwik
[26] Architectural Design Decision Support System 2.0, http://triana.escet.urjc.es/ADDSS
[27] Naval, F., Capilla, R., and Dueñas, J.C., Processes for Creating and Exploiting Architectural Design Decisions with Tool Support, Proceedings of the 1st European conference on Software Architecture (ECSA), pages 321–324, 2007.
[28] Liang, P., Jansen, and A., Avgeriou, P., Knowledge Architect: A Tool Suite for Managing Software Architecture Knowledge, RUG-SEARCH-09-L01, University of Groningen, 2009
[29] Liang, P., Jansen, A., and Avgeriou, P., A Collaborative Software Architecting Approach through Knowledge Sharing, Collaborative Software Engineering, Springer-Verlag, 2009
[30] Liang, P., Jansen, A., and Avgeriou, P., Refinement to Griffin Core Model and Model Mapping for Architectural Knowledge Sharing, RUG-SEARCH-07-L01, University of Groningen , 2007
[31] Tang, A., Han, J., and Vasa, R., Software Architecture Design Reasoning: A Case for Improves Methodology Support, IEEE Software, 26(2):43-49, 2009.
[32] Tang, A., Jin, Y., and Han, J., A Rationale-based Architecture Model for Design Traceability and Reasoning, Journal of Systems and Software, 80(6):918-934, 2006.
[34] Babar, M.A., and Gorton, I., A Tool for Managing Software Architecture Knowledge, Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK/ADI), pages 11-17, 2007.
[35] Babar, M.A., Gorton, I., and Kitchenham, B., A Framework for Supporting Architecture Knowledge and Rationale Management, In Rationale Management in Software Engineering, A.H. Dutoit, et al., eds, Springer, pages 237-254, 2006.
[36] Babar, M.A., Dingsøyr, T., Lago, P., and van Vliet, H., Software Architecture Knowledge Management: Theory and Practice, Springer-Verlag, 2009.
[37] Babar, M.A., de Boer, R.C., Dingsøyr, T., and Farenhorst, R., Architectural Knowledge Management Strategies: Approaches in Research and Industry, Proceedings of the 2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK/ADI), 2007.