Improving Understandability of Architecture Design through Visualization of Architectural Design Decision

Mojtaba Shahin
Department of Computer Engineering and IT
Sheikh Bahaei University, Iran
mojtabashahin@gmail.com

Peng Liang
State Key Lab of Software Engineering (SKLSE)
Wuhan University, China
liangp@sklse.org

Mohammad Reza Khayyambashi
Department of Computer Engineering
University of Isfahan, Iran
m.r.khayyambashi@eng.ui.ac.ir

ABSTRACT

Concentrating on components and connectors in traditional approaches to document software architecture causes the problems, such as high costs for architecture change and erode during architecture evolution. These problems result in a tendency to record architectural design decisions and their rationale made throughout architecting process. This tendency encourages practitioners and researchers to develop various models and related tools to model, capture, manage, share, and (re)use architectural design decisions. But there still remains a need to visualize and explore architectural design decisions due to the huge number of decisions and relationships among them in large and complex systems development. In this paper, we first make a survey on tools that support visualization of architectural design decisions, their features and deficiencies. Second we investigate how Compendium tool can be employed as a general tool to visualize architectural design decisions and their rationale. Last we present how the visualization by Compendium can improve the understandability and support the communication of architectural design in architecting process.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architecture—*Languages*; I.6.8 [**Computing Methodologies**]: Simulation and Modeling—*Visual*

General Terms

Design

Keywords

Software architecture, design rationale, architectural design decision, rationale visualization

1. INTRODUCTION

Software architecture plays an important role to mange complicated interactions between stakeholders of software-intensive systems in order to balance all kinds of constraints [2]. The architecting process can be considered as a decision-making process, through which the appropriate decisions should be made at the right time and place [4]. Traditional approaches to document software architecture concentrate on components and connectors which cause problems, such as expensive system evolution, lack of stakeholders communication, and limited reusability of architecture [6]. These problems are partially due to the volatilization of design decisions and their rationale made throughout architecting process [1], which are implicit in people's head mostly. Representation of design decision as a first-class entity in architecture design can partially alleviate these problems.

Practitioners and researchers have made great efforts to develop models and related tools to model, capture, manage, share, and (re)use architectural design decisions (ADDs) explicitly. In [11], the authors suggested five requirements for decision view in software architecture, in which one of them is visual representation of design decisions. Although there are many existing tools to capture, store, manage and share ADDs explicitly, there still remains a need to visualize and explore ADDs and their underlying rationale in architecting process [16].

Visualizing and exploring ADDs can assist the understanding of ADDs and the reasoning behind the design rationale, especially in a collaborative and distributed development environment. There are a few tools providing visualization support of ADD: Archium, Knowledge Architect (KA), Kruchten's ADD ontology tool, and Ontology-Driven Visualization tool (ODV). Archium supports the visualization of traceability between an ADD and related architecture artifacts [8]. Kruchten's ADD ontology tool focuses on the visualization of the dependencies and history of an ADD [13]. The KA tool suite emphasizes on visualizing the architectural knowledge entities and their relationships [14]. The main contribution of ODV tool is the analysis visualization of quality criteria to assist architecture auditors in decision-making process [3]. But none of them support explicit rationale visualization of an ADD. Compendium tool[1] is a semantic hypertext concept mapping tool that supports Issue-Based Information System (IBIS), an argumentation-based approach for design rationale representation. In this paper we first survey the advantages and deficiencies of the visualization support provided by existing ADD tools. Then

[1]http://compendium.open.ac.uk/institute/

we employ Compendium tool to visualize and explore ADDs in a concrete ADD example based on Service-Oriented Architecture Decision (SOAD) model [18], and show how rationale visualization capability of Compendium can address the deficiencies of existing ADD visualization tools.

The rest of this paper is organized as follows: Section 2 provides a survey and comparison on existing ADD visualization tools. Section 3 describes the mapping between IBIS notations and major ADD elements identified in our previous work [15], which provides a fundamental mapping framework for the mapping between concepts in Compendium and elements in various ADD models. In Section 4, we show how Compendium tool can be used to visualize and explore ADDs, and consequently support architecting activities with concrete examples. The initial evaluation results of Compendium tool are presented in Section 5. Section 6 describes a detailed plan for empirical validation of our proposed ADD visualization method and tool in controlled experiments. We conclude this paper with future directions in Section 7.

2. ADD VISUALIZATION TOOLS

Since Jansen and Bosch [6] proposed that software architecture can be viewed as composition of a set of architectural design decisions, many decision-centric architectural design tools have been developed. Some of these tools support users in visualizing and exploring ADDs. In this section, we make a brief survey and comparison on these tools.

2.1 Archium

Archium [8] is a design decision tool that aims to establish traceability between architectural design decision and other artifacts, such as *requirements*, other *decisions*, and *implementations*. In this tool, an ADD is regarded as a "change function" that modifies architecture design. Archium employs a component view to visualize ADDs and their relationships in a dependency graph. Every decision can be linked to architecture model (i.e., components and connectors). In this way, architects can select a component or connector in one view, and then Archium shows the related ADDs in another view. An example of this feature is shown in Figure 1.a.

2.2 Knowledge Architect

Knowledge Architect (KA) [14] is a tool suite for capturing, sharing, translating and managing software architectural knowledge (AK), in which ADD is an important type of AK. The KA tool suite comprises of six tools: Knowledge Repository, Document Knowledge Client, Excel Plug-in, Python Plug-in, Knowledge Explorer, and Knowledge Translator. The Knowledge Explorer provides various visualization supports to explore ADDs and their relationships. All the related entities of an ADD, such as *alternatives* and *requirements*, are visualized with forward and backward traceability link. In the Knowledge Explorer, users can easily inspect whether an ADD is (in)directly related to other ADDs of a specific type. Figure 1.b presents a screenshot of the Knowledge Explorer. A search function is provided to search AK entities (including ADDs) by AK types or keywords. The AK entity and its relationships with other AK entities are shown in the middle of the figure.

Table 1: A comparison of ADD visualization tools.

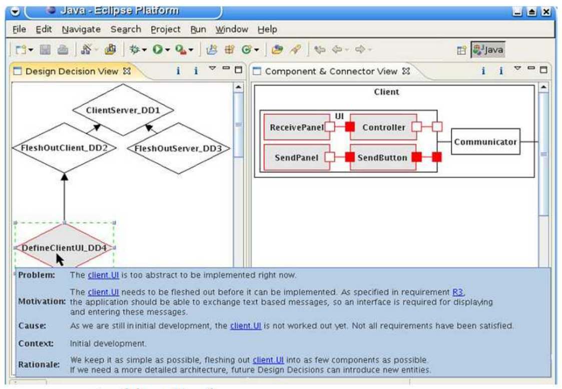
	Functionality					Architecting Support	
Existing Tools supporting ADD Visualization	Change impact analysis	Query support	Traceability support	Quantitative analysis	Collaborative architecting	Understandability of ADD	Modifiability of ADD
Archium	-	-	+	-	-	+	+
Knowledge Architect	+	+	+	+	+	++	++
Kruchten's ADD Ontology Tool	+	-	-	-	-	+	+
Ontology-Driven Visualization tool	+	-	-	+	-	+	+

2.3 Kruchten's ADD Ontology Tool

Lee and Kruchten [13] implemented a tool based on Kruchten's ADD ontology [10] to visualize ADDs. 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*". The goal of this tool is to support ADD exploration and analysis using four views: (1) 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) 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) decision chronology view, it supports a time-based view for showing the history of 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 a design decision during a specified time interval. (4) 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. A screenshot of this tool is shown in Figure 1.c.

2.4 Ontology-Driven Visualization Tool

The decision table and decision structure views introduced in Kruchten's ADD ontology tool have some drawbacks. A list or table does not show relationships effectively and also disregards much of the added value of using ontology [3]. However, decision structure view represents design decisions and their relationships impressively, but the resulting graph can become complicated for understanding when there are too many design decisions. de Boer et al. introduced an ontology-driven visualization (ODV) of ADDs that combines the strengths of table and structure visualization and overcomes their disadvantages. ODV tool can be used to assist software product audits, in which auditors are able to use it in two ways: (1) reusing their know-how (especially quality criteria) effectively; (2) supporting the core aspects of their decision-making process (especially trade-off analysis, impact analysis and "if-then" scenario). A screenshot of this tool is shown in Figure 1.d.



Problem: The client.UI is too abstract to be implemented right now.

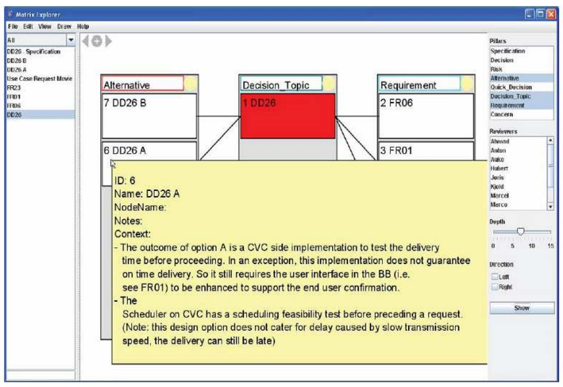
Motivation: The client.UI needs to be fleshed out before it can be implemented. As specified in requirement R3, the application should be able to exchange text based messages, so an interface is required for displaying and entering these messages.

Cause: As we are still in initial development, the client.UI is not worked out yet. Not all requirements have been satisfied.

Context: Initial development.

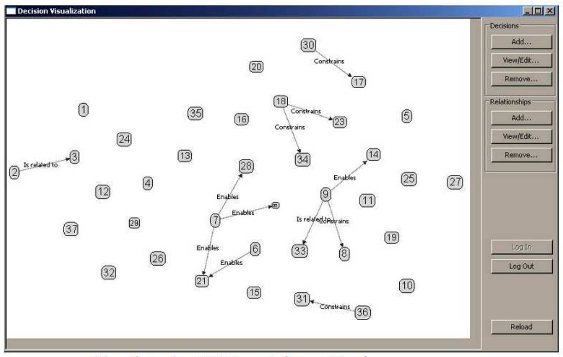
Rationale: We keep it as simple as possible, fleshing out client.UI into as few components as possible. If we need a more detailed architecture, future Design Decisions can introduce new entities.

a. Archium Tool

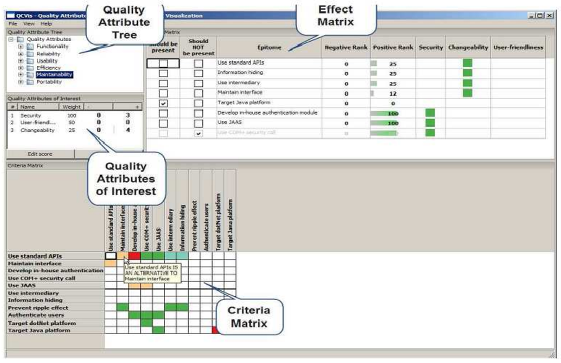


ID: 6
Name: DD26 A
NodeName:
Notes:
Context:
- The outcome of option A is a CVC side implementation to test the delivery time before proceeding. In an exception, this implementation does not guarantee on time delivery. So it still requires the user interface in the BB (i.e. see FR01) to be enhanced to support the end user confirmation.
- The Scheduler on CVC has a scheduling feasibility test before preceding a request. (Note: this design option does not cater for delay caused by slow transmission speed, the delivery can still be late)

b. Knowledge Architect (Knowledge Explorer)



c. Kruchten's ADD ontology Tool



d. Ontology-Driven Visualization Tool

Figure 1: Screenshots of ADD visualization tools.

2.5 Comparison of ADD Visualization Tools

In Table 1, a comparison is made on above ADD visualization tools in two levels: **functionality level** and **architecting support level**, in which "+" means "support", and "-" represents "no support". At the functionality level, we investigate what functions are provided by these ADD visualization tools. The criteria in this level include "change impact analysis", "query-support", "traceability support", "quantitative analysis", and "collaborative architecting", which are currently supported by some of these tools.

Functionality Level

Change impact analysis: This function shows which decisions or entities may be impacted by a change. The impact of changes can be visualized in the KA Knowledge Explorer. When users select a modified ADD, the KA Knowledge Explorer will visualize the related or potentially affected ADDs [14]. This feature is also supported by Kruchten's ADD ontology tool and ODV tool as well.

Query-support: The KA Knowledge Explorer provides a rich search function that users can query ADDs and related relationships.

Traceability support: Traceability means that the origins and targets of an ADD are traceable. In Archium and KA, users can create traces between AK entities, especially between ADDs and requirements.

Collaborative architecting: Multiple stakeholders exist in architecting process. They have various concerns, view-points, and backgrounds, in worst case, these stakeholders might be in a global and distributed setting. These stakeholders should work together, communicate, negotiate, and eventually agree upon the ADDs and design rationale. Among the ADD tools that support visualization, the KA can serve collaboration purpose by a central *knowledge hub* with various knowledge clients.

Quantitative analysis: Architecture offers the ability to predict expected quality of a software system before it is actually implemented or changed. One way to obtain such capability is quantitative analysis. The goal of quantitative analysis is to investigate how different design options or alternatives can satisfy one or more quality attributes of a software system [7]. AK and ADD of quantitative analysis are important for three needs: integration, verification, and validation of architecture design for quality attributes. The KA tool suite supports the quantitative analysis using ADDs and AK by two knowledge clients: the KA Excel Plug-in and Python Plug-in.

Architecting Support Level

At the architecting support level, we focus on how the ADD visualization tools can support the architecting process and activities, so this is in a higher level than the functionality level. The initial criteria in this level include "understandability of ADD" and "modifiability of ADD", which are also supported by these tools.

Understandability of ADD: Understandability refers to the capability that to what extent users with different backgrounds can understand the ADDs, or how new comers to the project are able to understand ADDs. All existing ADD visualization tools can support this feature. The KA

Knowledge Explorer improves the understanding of ADDs by visualizing the relationships among different AK entities, which is captured by the KA Document Knowledge Client, with additional rationale (e.g., *concerns*, *ranking*, etc.).

Modifiability of ADD: ADDs made throughout architecting process might need to be changed during architectural evolution. ADD visualization support can improve the modifiability of ADDs. When an ADD and its dependencies to other ADDs are visualized, the architect can easily estimate the work effort needed to change this ADD. The language used in Archium suggests explicit support for addition and modification of ADDs. In the KA tool suite, changes to AK can be edited using the KA Document Knowledge Client and Excel Plug-in [14].

According to the survey above, existing ADD tools for visualization have some deficiencies: (1) Archium and KA can not explicitly visualize dependency relationship between ADDs; (2) Collaborative decision-making is not supported by Archium, Kruchten's ADD ontology tool and ODV tool; (3) Rationale (e.g., *arguments*, *pros*, *cons*, and *decision derivers*) visualization support is missing in most of existing ADD tools. Archium, Kruchten's ADD ontology tool, and ODV tools can not visualize *alternatives*, *decision drivers*, and *justifications* of an ADD. These deficiencies can be fairly addressed by Compendium, a visualization tool based on IBIS (Issue-Based Information System) notations, which provides an argumentation-based approach for rationale representation, and visualizes the ADDs and their underlying rationale.

3. MAPPING BETWEEN IBIS NOTATIONS AND MAJOR ADD ELEMENTS

Issue-Based Information System (IBIS) provides a solution to tackle the wicked problems - complex, ill-defined problems that involve multiple stakeholders [12]. In recent years, there has been an increasing interest in using IBIS-type systems in software engineering [17], especially in design rationale and collaborative software development. IBIS consists of three basic and simple concepts: **Issues** (or *questions*) that need to be addressed; **Positions** (or *ideas*) that are responses to *questions*, typically are the set of *ideas* that respond to an *issue* representing the spectrum of perspectives on that *issue*; **Arguments** are composed of the **Pros** (arguments supporting) or **Cons** (arguments against) of a *position*. The three IBIS concepts are related with each other, and **Issue** is the core concept.

In [15], we analyzed nine ADD tools, and classified the ADD elements in two types: major elements and minor elements. Major elements include *Problem*, *Constraint*, *Solution*, *Decision*, and *Rationale* that all ADD models have a consensus on. *Group*, *Status*, *Dependency*, *Artifact*, *Consequence*, *Stakeholders*, and *Phase/Iteration* are elements without a consensus, called minor elements. Table 2 shows the mapping between the three basic IBIS concepts and some of the major ADD elements. This conceptual mapping between IBIS concepts and ADD major elements establishes a bridge between rationale tools based on IBIS and ADD tools based on ADD major elements. However, there is no corresponding IBIS concept to two ADD major elements: *Decision* and *Constraint*. But it is possible to map these two ADD major elements to Compendium tool that extends the IBIS concepts.

Table 2: Mapping between part of ADD major elements and IBIS concepts.

ADD Major Elements	IBIS Concepts
Problem	Issue (question)
Solution	Position (idea)
Rationale	Arguments (Pros and Cons)

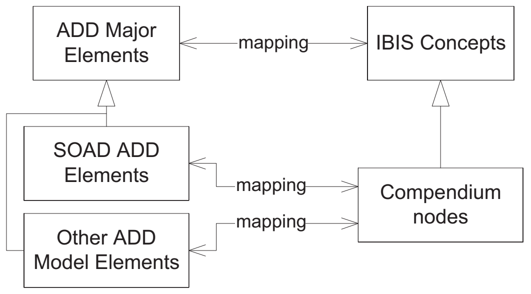


Figure 2: The generalization of mapping between ADD major elements and IBIS concepts.

Compendium is an open source software tool that is conceptually implemented based on IBIS and supports graphical IBIS notations. Compendium visually represents thoughts and illustrates various interconnections between *ideas* and *arguments*. The "issue maps" created by Compendium graphically represent the relationships between *issues*, and facilitate the understanding of inter-connected topics, through visual representation. Compendium can also be used by a group of people in a collaborative manner to convey ideas to each other through the use of visual images. In Section 4, we show how to employ Compendium, a general design rationale visualization tool, to visualize ADDs and their underlying rationale. Note that the Knowledge Explorer (shown in Figure 1.b) in the KA tool suite that we have developed for visualization of AK entities, has no direct relationship with this Compendium-based ADD visualization tool since they try to address different visualization features: the Knowledge Explorer targets to the visualization of general AK entities, and Compendium emphasizes on rationale visualization of ADDs.

4. RATIONALE VISUALIZATION OF ADD

We adopt an ADD model on service-oriented architecture design, the SOAD model [18], and visualize and explore the ADDs based on this model using Compendium. It is worth noting that one of the major contribution of this paper is the mapping we create between IBIS concepts and ADD major elements. As mentioned in previous section, this mapping establishes the bridge between tools based on IBIS (e.g., Compendium) and ADD tools based on ADD major elements, since Compendium is based on/extended from IBIS concepts and all ADD tools have a consensus on ADD major concepts [15]. So not only SOAD ADDs can be visualized by Compendium, ADDs based on other ADD models can also be visualized by Compendium (as shown in Figure 2).

The ADD concepts in SOAD model is shown in Figure 3. In this model, *Architectural Decision (AD)* is the central entity that expresses a single and concrete design decision. This entity has characteristics like *Scope*, *Role*, *De-*

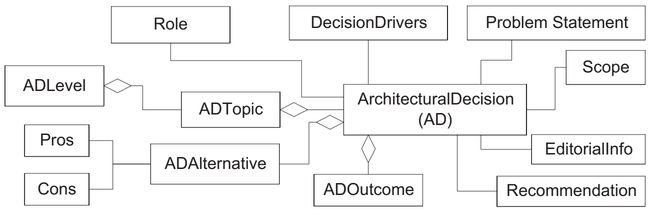


Figure 3: SOAD architectural design decision model.

Table 3: SOAD concept mappings in Compendium.

SOAD Concepts	Compendium Nodes	
Architectural Decision (AD)	●	Decision
Role, Scope, EditorialInfo, Decision Drivers, Recommendation	●	Note
Problem Statement	?	Question
ADAlternative	💡	Answer
Pros	⊕	Pro Argument
Cons	⊖	Con Argument

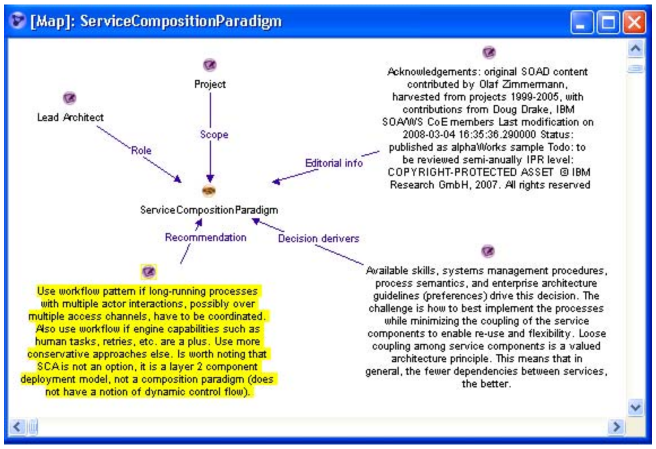
cision Drivers, *Problem Statement*, *Recommendation*, and *Editorialinfo*. In this model, every *AD* has one or several *AD alternatives* with specification of *Pros* and *Cons*.

In this model, ADDs can be categorized in three levels: Conceptual level, Technology level and Vendor level, and all *AD topics* reside on one of these *AD levels*. A tool AD_{kwik}[2] (Architectural Decision Knowledge Wiki), as a collaboration system for ADD management, is implemented to support the SOAD model. This tool can capture, store, manage, and share ADDs and their rationale in a text-based manner, but does not provide any visualization capability. We employ Compendium to visualize SOAD ADDs.

4.1 Mapping from SOAD to Compendium

In this section, we discuss how SOAD concepts are mapped to Compendium nodes based on the mapping between ADD major elements and IBIS concepts in Table 2. The nodes in a Compendium map are labeled as *Decision*, *Note*, *Question*, *Answer*, and *Argument* (*Pro Argument* and *Con Argument*). Table 3 presents the mapping between Compendium nodes and SOAD concepts. *Decision* node in Compendium can be mapped to *Architectural Decision (AD)* concept in SOAD model. *Note* node in Compendium can be used to provide extra and useful information about a node, therefore we can map *Note* node to *Role*, *Scope*, *EditorialInfo*, *Decision Drivers*, and *Recommendation* concepts. *Question* node in Compendium can be used to represent *Problem Statement* concept in SOAD. *Answer* node in Compendium represents alternative solutions addressing to *Question*, *Pros* and *Cons*, which support and object to this answer respectively. Every design decision can have one or more alternatives, and therefore *Answer* node is used to represent *ADAlternative* concept in SOAD model. *Pro* and *Con Argument* can be directly mapped to *Pros* and *Cons* in SOAD.

[2]http://www.alphaworks.ibm.com/tech/adkwik



Acknowledgements: original SOAD content contributed by Olaf Zimmermann, harvested from projects 1999-2005, with contributions from Doug Drake, IBM SOA/WS CoE members Last modification on 2008-03-04 16:35:36.290000 Status: published as alphaWorks sample Todo: to be reviewed semi-anually IPR level: COPYRIGHT-PROTECTED ASSET © IBM Research GmbH, 2007. All rights reserved

Use workflow pattern if long-running processes with multiple actor interactions, possibly over multiple access channels, have to be coordinated. Also use workflow if engine capabilities such as human tasks, retries, etc. are a plus. Use more conservative approaches else. Is worth noting that SCA is not an option, it is a layer 2 component deployment model, not a composition paradigm (does not have a notion of dynamic control flow).

Available skills, systems management procedures, process semantics, and enterprise architecture guidelines (preferences) drive this decision. The challenge is how to best implement the processes while minimizing the coupling of the service components to enable re-use and flexibility. Loose coupling among service components is a valued architecture principle. This means that in general, the fewer dependencies between services, the better.

Figure 4: Visualization of SOAD ADD attributes in Compendium.

4.2 SOAD ADD Visualization Examples

We use an example from `SoadSample3` provided by AD_{kwik} to visualize SOAD ADDs. Figure 4 and 5 show how an ADD (e.g., *ServiceCompositionParadigm*) in SOAD model can be visualized in Compendium. Figure 4 shows how the attributes of an ADD in SOAD model can be visualized by Compendium. The *scope* and *role* of this ADD are "*Project*" and "*Lead architect*" respectively that are represented by *Note* (●) node in Compendium.

Figure 5 presents the visualization of a SOAD ADD and its related entities in Compendium. The design *problem* (?) to be addressed is "*Should a Service Composition Layer (SCL) be introduced into the solution architecture?*". The "*ServiceCompositionParadigm*" *decision* (●) gets selected with five *alternatives* (💡) "*Integration layer*", "*SCL and workflow pattern*", "*Programming language component model*", "*Portal or mashup*", and "*human user invoking atomic business functions*". Every *alternative* has positive and negative effects to the design *problem*, which are represented by *Pro* (⊕) and *Con* (⊖) *Argument*[3]. The rationale of an ADD can be easily visualized using Compendium in a straightforward way. For example, the positive effect (⊕) of *alternative* (💡) "*Programming language component model*" is that it is a "mature technique", while the negative effect (⊖) of this *alternative* is due to its "*not leveraging SOA benefits*". Users can easily follow the visualization link between rationale entities through an ADD in Compendium, and update their decision when design context changes (e.g., "*when a component model is employed instead of SOA*", then the negative effect "*not leveraging SOA benefits*" can be ignored/removed).

In SOAD model, ADDs might have dependencies with other ADDs, including *influences*, *decomposesInto*, *refinedBy*, *forces*, *isIncompatibleWith*, *triggers*, and *prunes*. Figure 6 demonstrates the capability of Compendium for visualizing the dependencies between ADDs, for example *ServiceCompositionParadigm* decision has a dependency with *ServiceCompositionLanguage*, *ResourceProtectionStrategy*, *TransactionCoordinator*, and *CompensationMechanism* ADDs that all of them have an *influences* relationship with *ServiceCompositionParadigm*. Figure 6 shows how Compendium can

[3]The *decision* also has *pro* and *con arguments*, which are not shown in Figure 5 due to the space limitation.

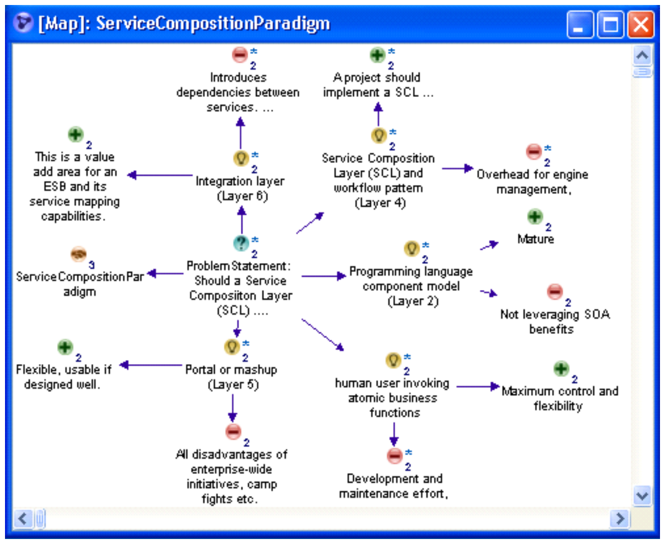

[Map]: ServiceCompositionParadigm
Introduces dependencies between services. ...
A project should implement a SCL ...
This is a value add area for an ESB and its service mapping capabilities.
Integration layer (Layer 6)
Service Composition Layer (SCL) and workflow pattern (Layer 4)
Overhead for engine management.
Mature
ServiceCompositionParadigm
Problem Statement: Should a Service Composiiton Layer (SCL) ....
Programming language component model (Layer 2)
Not leveraging SOA benefits
Flexible, usable if designed well.
Portal or mashup (Layer 5)
human user invoking atomic business functions
Maximum control and flexibility
All disadvantages of enterprise-wide initiatives, camp fights etc.
Development and maintenance effort.

Figure 5: Visualization of a SOAD ADD and related entities in Compendium.

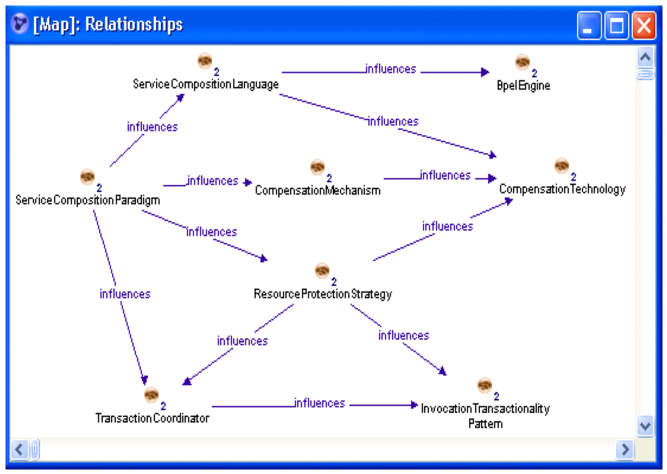


Figure 6: Visualization of SOAD ADD dependencies in Compendium.

visualize the dependencies between ADDs by labeling the connections between them. Note that, Compendium can specify any relationships other than *influences* relationship, such as *forces*, *refinedBy*, etc., and any user-specific relationships, but in the `SodaSample3` example, the ADDs only have *influences* relationship with each others.

4.3 Improving Understandability of Architecture Design

Architecting process is composed of three main activities: architectural analysis, architectural synthesis and architectural evaluation [5]. In architectural analysis, architects select architecturally significant requirement (ASR) from architectural concerns and context, and then architects propose architectural solutions (*alternatives*) to address a set of ASRs in architectural synthesis. In architectural evaluation, architects evaluate the proposed solutions and make a design decision by selecting among and trading off candidate solutions. According to the examples presented in Section 4.2, rationale visualization of ADD by Compendium can assist architecting process through the three architecting activities in the following ways:

Architectural analysis: architects select and analyze an ASR, e.g., "*Should a Service Composition Layer (SCL) be introduced into the solution architecture?*", "*Should process orchestration be used to implement cross-service component business process logic?*", which are original from architectural concerns and context, e.g., "*available technical skills*" and "*the fewer dependencies between services, the better*". Figure 4 (*decision drivers*) and Figure 5 (*problem statement*) show how Compendium can visualize this activity. This kind of visual information can assist architect to understand the root cause of ASRs and further design.

Architectural synthesis: in this activity, architects propose several architectural solutions, e.g., "*Integration layer (Layer 6)*" and "*Programming language component model (Layer 2)*", etc., to address the ASRs obtained in previous activity. As we have shown in Figure 5, architects can use Compendium to visualize these architectural solutions using *Answer* (💡) node. The design rationale of an ADD can be visualized in this activity by adding *pro* (➕) and *con* (➖) *arguments* to each architecture candidate solution, i.e., *Answer* (💡), so that these solutions can be easily understood and evaluated in the next activity.

Architectural evaluation: in this activity, architects evaluate candidate solutions based on the *pros* and *cons* of each solution, and choose one. When an ADD is made, architects can also visualize the dependencies between this decision and other ADDs, so that architects can evaluate architectural design integrally, especially in a complex architecture design context, the decisions are heavily influenced by each other. For example, when *ServiceCompositionParadigm* decision is made, architects can visualize that this decision has an *influences* relationship with other ADDs, e.g., *ServiceCompositionLanguage*. Figure 6 shows how Compendium can visualize dependencies between ADDs. Architects can easily understand the impact of an ADD through this visualization when an ADD is modified (selected or rejected).

5. EVALUATION OF COMPENDIUM TOOL

In Table 4, Compendium is evaluated according to the criteria stated in Table 1 to demonstrate the effectiveness of Compendium, as a general rationale visualization tool, for ADD visualization and other ADD tool capabilities.

Change impact analysis: As we have shown in Figure 6, Compendium can visualize the dependencies between ADDs. Through the dependency view of a decision with other decisions, users can figure out/understand the intensity/weight (how important) of an ADD by these dependencies (e.g., number and type of dependencies). Users can also analyze what decisions and how they are impacted by a change of a decision through the dependency links. The ODV tool also well supports this feature among existing ADD tools.

Query-support: Compendium provides a search function that users can search the entities of ADD by keyword, and filters the results by tags, node types, date and authors, etc.

Traceability support: As shown in Figure 4 and 5,

Table 4: Compendium evaluation based on the criteria in Table 1.

Existing Tools supporting ADD Visualization	Functionality					Architecting Support	
	Change impact analysis	Query support	Traceability support	Quantitative analysis	Collaborative architecting	Understandability of ADD	Modifiability of ADD
Compendium Tool	+	+	+	-	+	++	++

Compendium provides traceability support among requirements (*Questions*), *decisions*, and alternatives (*Answers*) of a decision.

Collaborative architecting: Compendium supports multiple users with a visual environment to facilitate collaborative decision-making process. Since Compendium can document and visualize design rationale behind ADDs, it can better support collaborative architecting through rationale communication among multiple users.

Quantitative analysis: Compendium currently can not support this feature.

Understandability and Modifiability of ADD: In general, rationale visualization of ADDs using Compendium can improve the quality of architecture design in several ways. It can assist software architects for better understanding of ADDs and underlying rationale, which consequently results in better understanding of architecture documentation. It can also promote the communication of architecture documentation by visualization of *alternatives*, *decision drivers*, and *arguments* of ADDs and their dependencies, especially when the number of ADDs and their *alternatives* increases, consequently speeds up related architecting activities. Visualizing various types of dependency relationships between ADDs in Compendium results in better understanding of change impact when architects want to add, change, or remove ADDs. Visualization of ADDs and their rationale can also speed up the learning process for new architects of a project because they can easily review the rationale of previously-made ADDs, and understand architecture design more quickly and reliably.

6. PLANNED CONTROLLED EXPERIMENTS

For the credibility of our proposed ADD visualization method with Compendium tool support, we plan to empirically validate our claim that the rationale visualization of ADD using Compendium can improve the understandability of architecture design. We present the objective of this controlled experiment according to the template suggested in [9]: *Analyze* Compendium as an ADD visualization tool *for the purpose of* improving (*with respect to*) the understandability of architecture design *from the point of view of* (experimental **Subject**) *in the context of* (experimental **Object**).

Research Questions and Hypotheses

In the controlled experiments, we plan to compare the understandability one has of architecture design when using Compendium for visualizing ADD and their rationale as opposed to without using it. To this end, we need a way to quantify the understandability someone has of architecture design. Because understanding architecture design is crucial for architect's ability to perform architecting activities, we can indirectly measure the understandability architect has of architecture design by evaluating how well he/she performs architecting activities in architecting process. Based on this assumption about the relationship between understandability and architecting activities, the following research questions are formulated:

- *Q1*: Does using Compendium for visualizing ADDs and their rationale reduce the *time* effort that is needed to understand architecture design in architecting process against without using it?
- *Q2*: Does using Compendium for visualizing ADDs and their rationale increase the *correctness* of understanding architecture design in architecting process as opposed to without using it?

Associated with these two research questions are two null hypotheses formulated as follows:

- $H1_0$: Using Compendium for visualizing ADDs and their rationale does not impact the *time* needed to understand architecture design in architecting process.
- $H2_0$: Using Compendium for visualizing ADDs and their rationale does not impact the *correctness* of understanding architecture design in architecting process.

Subject: Selected persons (e.g., specific users group) who are going to participate in the experiments.

Object: The system (i.e., for which an architecting process will be employed to design the architecture) considered to perform the experiments.

The experiment is conducted by two groups A and B. The group A and B use the same architecture document (*Object*) to perform the experiments. The group A uses the architecture document without the support of rationale visualization capability, and group B performs the experiment on the same document with rationale visualization supported by Compendium. The experimental data (time consumption and understanding results) collected from the two groups is analyzed and compared in order to investigate and evaluate whether the hypotheses are true or not. The controlled experiments are planed to be conducted in the next step.

7. CONCLUSIONS AND FUTURE WORK

Software architects are particularly interested in understanding why the software architecture looks the way it does. The architectural design decisions and their rationale can answer this question when they are captured, managed, shared, and (re)used decently and explicitly. Although many ADD models have been proposed and related tools have been developed for these objectives, there is still a need to visualize and explore ADDs and their underlying rationale. In this paper, we employ Compendium, a general and ready-to-use rationale visualization tool, to visualize ADDs and the underlying rationale. Visualization of ADD using Compendium can assist architects in the following points: (1) improve the understandability of ADD and their rationale that leads to better understanding of architecture design; (2) promote the communication of architecture documentation by visualization of alternatives, decision drivers, and arguments of ADDs; (3) speed up architectural evaluation activity by facilitating review and change impact analysis of ADDs.

There are also several limitations in this work: (1) not all the ADD model elements can be one-to-one mapped to Compendium nodes, and currently we employ the *Note* node to describe these unmappable ADD elements as an alternative mapping solutions in order to show these ADD elements in Compendium, for example, the SOAD concepts *Role*, *Scope*, *EditorialInfo*, *Decision Driver*, and *Recommendation* are mapped to *Note* node. Better solutions should be provided in order to well present the meaning of and relationships between various ADD elements. (2) Design rationale visualization helps in architectural analysis and explanation, but in other situations, especially in a creative industry, architects might like to synthesize and build architecture design based on their understanding of rationale. So how improved understandability of architecture design can help to reuse and lead to the reusable architecture design, is also a limitation which is not covered in this paper.

Besides the limitations, we outline our ongoing and future work in the following points: (1) validate experimentally our claims on improved understandability of architecture design using Compendium with controlled experiments as discussed in Section 6; (2) investigate how rationale visualization of ADD can systematically support and improve the architecting process, including other architecting activities (e.g., architectural implementation and maintenance); (3) partially automate the visualization support of ADDs, for example, when users select an ADD documented in text, the visualization tool can automatically extract related entities (e.g., problem, alternatives, pros and cons) and visualize them.

8. ACKNOWLEDGMENTS

This research has been partially sponsored by the Natural Science Foundation of China (NSFC) under Grant No.60903034, QuASAK: Quality Assurance in Software architecting process using Architectural Knowledge. We also would like to thank all the reviewers for their constructive comments.

9. REFERENCES

[1] M. Babar and P. Lago. Design Decisions and Design Rationale in Software Architecture. *The Journal of Systems & Software*, 82(8):1195–1197, 2009.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, 2nd edition*. Addison-Wesley Professional, 2003.

[3] R. de Boer, P. Lago, A. Telea, and H. van Vliet. Ontology-Driven Visualization of Architectural Design Decisions. In *Proceedings of the 8th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 43–52. IEEE Computer Society, 2009.

[4] R. Farenhorst, P. Lago, and H. van Vliet. EAGLE: Effective Tool Support for Sharing Architectural Knowledge. *International Journal of Cooperative Information Systems*, 16(3/4):413–437, 2007.

[5] C. Hofmeister, P. Kruchten, R. Nord, H. Obbink, A. Ran, and P. America. A General Model of Software Architecture Design derived from Five Industrial Approaches. *The Journal of Systems & Software*, 80(1):106–126, 2007.

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

[7] A. Jansen, T. de Vries, P. Avgeriou, and M. van Veelen. Sharing the Architectural Knowledge of Quantitative Analysis. In *Proceedings of the 4th International Conference on the Quality of Software-Architectures (QoSA)*, pages 220–234. Springer LNCS, 2008.

[8] A. Jansen, J. Van Der Ven, P. Avgeriou, and D. Hammer. Tool Support for Architectural Decisions. In *Proceedings of the 6th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 44–53, 2007.

[9] A. Jedlitschka and D. Pfahl. Reporting Guidelines for Controlled Experiments in Software Engineering. In *Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE)*, pages 95–104, 2005.

[10] P. Kruchten. An Ontology of Architectural Design Decisions in Software Intensive Systems. In *Proceedings of the 2nd Groningen Workshop on Software Variability Management (SVM)*, pages 54–61, 2004.

[11] P. Kruchten, R. Capilla, and J. C. Dueñas. The Decision View's Role in Software Architecture Practice. *IEEE Software*, 26(2):36–42, 2009.

[12] W. Kunz and H. Rittel. *Issues as Elements of Information Systems*. Studiengruppe fur Systemforschung, 1970.

[13] L. Lee and P. Kruchten. A Tool to Visualize Architectural Design Decisions. In *Proceedings of the 4th International Conference on the Quality of Software Architectures (QoSA)*, pages 43–54, 2009.

[14] P. Liang, A. Jansen, and P. Avgeriou. Knowledge Architect: A Tool Suite for Managing Software Architecture Knowledge. Technical Report RUG-SEARCH-09-L01, University of Groningen, 2009.

[15] M. Shahin, P. Liang, and M. Khayyambashi. Architectural Design Decision: Existing Models and Tools. In *Proceedings of the 8th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 293–296. IEEE Computer Society, 2009.

[16] M. Shahin, P. Liang, and M. R. Khayyambashi. A Survey of Architectural Design Decision Models and Tools. Technical Report SBU-RUG-2009-SL-01, Sheikh Bahaei University & University of Groningen, 2009.

[17] S. Shum, A. Selvin, M. Sierhuis, J. Conklin, C. Haley, and B. Nuseibeh. Hypermedia Support for Argumentation-based Rationale: 15 years on from gIBIS and QOC. In *Rationale Management in Software Engineering, A.H. Dutoit, et al., (Eds.)*, pages 111–132. Springer, 2006.

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