Towards Automated Solution Synthesis and Rationale Capture in Decision-Centric Architecture Design

Xiaofeng Cui, Yanchun Sun*, Hong Mei
Institute of Software, School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing 100871, China
{cuixf04, sunyc}@sei.pku.edu.cn, meih@pku.edu.cn

Abstract

Software architectures are considered crucial because they are the earliest blueprints for target products and at the right level for achieving system-wide qualities. Existing methods of architecture design still face the challenge of bridging the gap between software requirements and architectures in practice. The emerging methods that focus on design decisions and rationale provide little support for deriving target architectures. In this paper we propose a decision-centric architecture design approach, which models issues, solutions, decisions, and rationale as the core elements of architecture design and the key notions to direct the derivation of target architectures. The approach transits from requirements to architectures through a process including issue eliciting, solution exploiting, solution synthesizing, and architecture deciding. We implement the automated synthesis of candidate architecture solutions from various issue solutions, and provide a way to capture comprehensive design decisions and rationale during this design process. We finally illustrate the applicability of this approach with a case study.

1. Introduction

Software architectures are considered crucial because they are the earliest blueprints for target products and at the right level for achieving system-wide qualities. Architecture design therefore plays a decisive role in the whole lifecycle of software. A number of methods have been proposed for this purpose [1]. Most of the methods provide step-by-step processes and strategic guidelines to accomplish architecture design. Because of the inherent gap between software requirements and architectures, there is still a challenge of providing pragmatic assistant for practitioners to cope with the difficulty and complexity of architecture design.

Recently, software architecture community has an emerging focus on design decisions and rationale [2]. Many methods have been proposed for the representation and usage of architecture decisions and rationale. However, these notions by now provide little support for deriving target architectures. On the other hand, although the importance of recording design rationale is widely recognized, the success of the methods lies in how to accomplish this task efficiently. The overhead of capturing and recording rationale and the interference with the architecting process may impair the expected benefits.

It is well accepted that software architectures manifest the earliest design decisions [3]. We emphasize that the essential task of architecture design is the decision-making on the key problems that are situated at architecture level and have dominant impacts on subsequent detailed design. To each of these architectural problems, multiple candidate solutions may emerge from various stakeholders and based on various design knowledge, technologies, experiences, etc. Each solution usually has its own advantages and disadvantages, and these solutions are usually interdependent and intertwining. This situation is the main cause of the difficulty in architecture decision-making. To select the solution for each problem, it is necessary to analyze its global impacts and balance it against other related problems. In our opinion, the rational decisions cannot be made without comprehensively considering the architecture as a whole. For this reason, we argue that architects need to firstly exploit the possible architectures that are made up of the candidate solutions to every distinct problem,

[*] Corresponding author

and then make decisions based on the evaluation and comparison of these candidate architectures. When the ultimate architecture is selected, the architecture decision is made and the decisions on every distinct problem are therefore settled. During this decision-centric design process, it is definitely a nontrivial effort to synthesize candidate architectures systematically, so in practice many resulting architectures are derived on an ad hoc basis.

In this paper we propose a decision-centric architecture design approach, which models issues, solutions, decisions, and rationale as the core elements of architecture design and the key notions to direct the derivation of target architectures. The approach transits from software requirements to architectures through a process including issue eliciting, solution exploiting, solution synthesizing, and architecture deciding. We implement the automated synthesis of candidate architecture solutions from various issue solutions, so that a whole solution space of feasible architectures can be presented in order for architects to explore and make decisions. We also provide a way to capture comprehensive design decisions and rationale during the design process, so that this knowledge can be recorded integrally and retrieved easily for the comprehension and communication of architectures.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 gives an overview of the decision-centric meta-model and architecture design process. Section 4 describes the synthesis of architecture solution. Section 5 describes the capture of design rationale. Section 6 illustrates the approach with a case study. Section 7 discusses the contribution and applicability of this approach. Finally, Section 8 presents concluding remarks and future work.

2. Related work

2.1 Software architecture design methods

A number of methods have been proposed for software architecture design. Many methods derive the resulting architecture via a series of transformations and provide guidelines for certain steps. QASAR [4] employs iterative evaluation and transformation of software architecture in order to satisfy non-functional requirements (NFRs). The method also formulates design guidelines to indicate suitable transformations. Quality-Driven Composition (QDC) [5] firstly derives the architecture that only addresses functional requirements and contains a number of variability points. The solution fragments addressing quality requirements are used to iteratively compose the architecture. Attribute-Driven Design (ADD) [3] follows a recursive process that decomposes a system or system element by applying architectural tactics and patterns that satisfy its driving quality attribute requirements. Siemens Four-Views (S4V) [6] proposes a Global Analysis where architects identify architectural issues, propose design strategies to solve the issues, and apply them to one or more of the views. In RUP's 4+1 View method [7], architectural design spreads over several iterations in an elaboration phase, iteratively populating the four views, driven by architecturally significant use cases, nonfunctional requirements, and risks.

Some other methods focus on making choice among candidate architectures via evaluation and decision-making process. It is a prerequisite for these methods to achieve the candidate architectures beforehand. For example, Svahnberg et al. [8] propose a quality-driven decision-support method for identifying software architecture candidates. AI-Naeem et al. [9] use optimization techniques to recommend the optimal candidate architecture.

Hofmeister et al. [1] compare five industrial software architecture design methods and extract a general model, which classifies the kinds of activities performed during design into architectural analysis, architectural synthesis, and architectural evaluation. Architectural analysis articulates architecturally significant requirements (ASRs). Architectural synthesis results in candidate architectural solutions. Architectural evaluation ensures that the architectural decisions used are the right ones.

Our approach has similarities to existing methods from the perspectives of employing iterative process, addressing architecturally significant requirements, etc. This approach also complies with Hofmeister et al.'s general model of architecture design. Our emphasis is to provide pragmatic support for deriving target architectures. We achieve this goal by means of automated synthesis of candidate architecture solutions from which architects can make their selection, whereas few existing methods facilitate this task effectively. In addition, our architecture design process accommodates many existing methods, e.g., the Global Analysis of S4V for eliciting architectural issues and the multiple-criteria decision-making of Svahnberg's method for selecting the target architecture.

2.2 Architecture design decisions and rationale

The software architecture community has paid attention to design decisions and rationale for a long time. Perry and Wolf [10] present a model of software architecture that consists of three components: elements, form, and rationale. Bosch [2] promotes that design decisions should be represented as first-class entities in software architectures, and a software architecture is fundamentally a composition of architectural design decisions. Tyree and Akerman [11] claim that a key to demystifying architecture products lies in the architecture decisions concept. Kruchten et al. [12] propose an ontology of software design decisions. Jansen et al. [13, 14] give a meta-model of software architecture decisions and develop a tool to model the software architecture as the composition of design decisions. Tang et al. [15] introduce the rationale-based model, Architecture Rationale and Elements Linkage (AREL), to support design rationale capture and traversal.

Although the importance of architecture design decision and rationale has been widely recognized, these notions provide little support for deriving target architectures by now. Existing methods mostly focus on the modeling of design decisions and rationale, their integration into architecture models, and the usage and reuse of design decisions for architecture comprehension. Akerman and Tyree [16] pay attention to the development of architecture. They propose five steps, i.e., capturing stakeholder concerns, analyzing the current architecture, defining the target architecture, conducting a gap analysis and producing the roadmap, and validating the architecture, to develop their architectural ontology.

Our approach not only models the architecture design decisions and rationale, but also provides support for accomplishing the design process. Our meta-model distinguishes between the issue-level and architecture-level notions of solution, decision, and rationale. This hierarchical meta-model is the conceptual foundation for the automated synthesis of architecture solutions from issue solutions, and the deduction of issue decisions and rationale from architecture decisions and rationale.

3. The decision-centric meta-model and architecture design process

3.1 Decision-centric meta-model

We propose a meta-model to describe the basic notions in our approach of decision-centric architecture design. Figure 1 shows these notions and their relationships to software requirements (RE) and software architecture (SA).

An *issue* is an architecturally significant requirement [1], which addresses one specific aspect of requirements or any other high level considerations. Issues act as a linkage between the requirements, which are naturally articulated from customers' perspectives, and the designs, which are always considered from architects' perspectives.

The notion of *solution* is classified into *issue solution* and *architecture solution*. An issue solution provides a possible way of solving an issue. An architecture solution is a candidate architecture design that has addressed all of the issues. The architecture solutions can be synthesized from the issue solutions. A solution contains its description, pros, cons, etc.

The notion of *decision* is classified into *issue decision* and *architecture decision*. An issue decision means adopting or discarding one candidate issue solution. An architecture decision means adopting or discarding one candidate architecture solution. Issue decisions can be determined by the architecture decision. A decision contains its description, etc.

The notion of *rationale* is classified into *issue rationale* and *architecture rationale*. Issue rationale is the reason behind issue decisions. Architecture rationale is the reason behind architecture decisions. Issue rationale can be deduced from the architecture rationale. Rationale contains its description, etc.

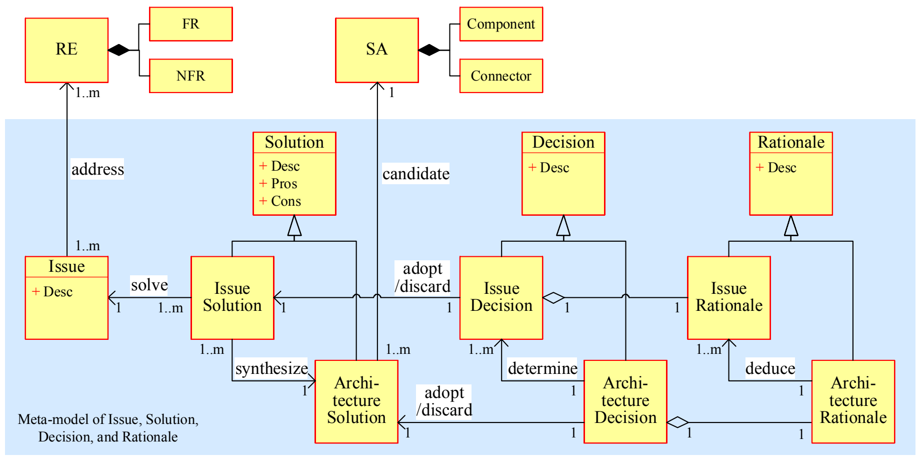

Figure 1. The decision-centric meta-model

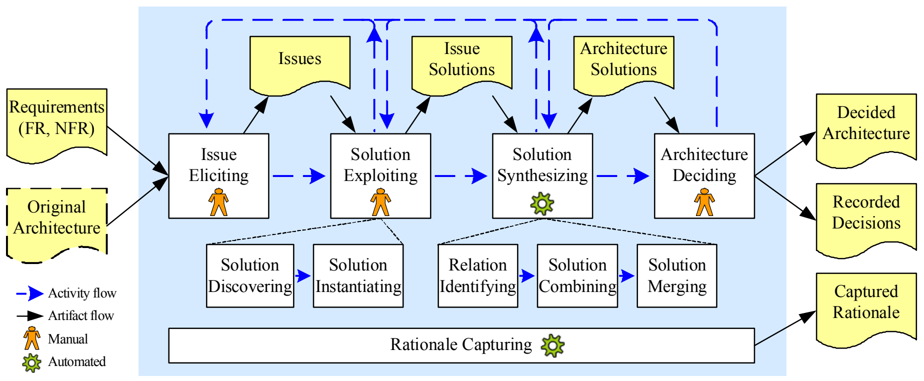

Figure 2. The decision-centric architecture design process

3.2 Decision-centric design process

We propose an iterative process to implement our decision-centric architecture design. The main inputs to this process are software requirements and original architecture. The main outputs of this process are decided architecture, recorded decisions, and captured rationale. The process has four major successive activities: issue eliciting, solution exploiting, solution synthesizing, and architecture deciding. There is also a rationale capturing activity spreading over the process. Figure 2 shows this architecture design process.

An original architecture represents preliminarily determined skeleton at the start of our design process. It is not a necessary input to the process. If exists, it can act as a basis to elicit the issues as well as a foundation of architecting. Otherwise the design process can start from scratch.

In the issue eliciting activity, stakeholders (e.g., architects, customers, etc.) deliberate and determine architecturally significant issues, based on the requirements, the original architecture, and other global considerations, e.g., project constraints, available COTS, etc. The subsequent activities may also elicit new issues. Therefore the whole process will be iterated until no new issue comes forth.

In the solution exploiting activity, architects derive candidate solutions to each issue. First, they discover solutions, according to reusable design knowledge or newly developed technologies. Second, they instantiate these solutions from informal descriptions to concrete design fragments modeled with architecture elements. The solution exploiting activity may be revisited from subsequent activities for the purpose of refining.

The solution synthesizing activity automatically synthesizes the candidate architecture solutions from various issue solutions. First, the relations between issue solutions are identified to indicate whether they can be combined together. Second, all feasible combinations of issue solutions are explored. Finally, the issue solutions within each feasible combination are merged to generate the architecture solutions.

The last activity is architecture deciding, in which stakeholders select the target architecture solution from the synthesized candidate architectures and validate them. To make the selection, they need to evaluate the candidate architecture solutions according to the requirements, compare them by multiple criteria, and make trade-offs between the competing objectives.

The rationale capturing activity can automatically deduce issue decisions and rationale from the settled architecture decision and rationale. This is accomplished based on the synthesis relationship between the architecture solutions and issue solutions. The deduced issue decisions and rationale, as well as the architecture decision and rationale which are mainly made and specified by people, constitute an integral set of design decisions and rationale.

This process provides a workflow that can achieve practical architecture design and a context where the automated solution synthesis and rationale capture can be realized. The following two sections describe how architecture solutions can be synthesized and how design rationale can be captured automatically. The activities of issue eliciting and architecture deciding are not the focus of this paper.

4. The Synthesis of architecture solutions

4.1 Solution exploiting

In our approach, architects accomplish the exploiting of issue solutions via two successive steps.

(1) Solution Discovering

For each elicited issue, the candidate solutions may be discovered in several ways. Primarily, architects proposed solutions based on their expertise and experience. They can also leverage the codified design knowledge, e.g., design patterns and architecture patterns [17]. Additionally, the solutions may originate from legacy systems, new technologies, commercial products, etc.

For example, to the issue "Convenient information acquiring in enterprise application", two candidate solutions can be discovered: "Using a C/S structure" or "Using a B/S structure".

The pros and cons of each issue solution are also specified here. If one solution is derived from codified design knowledge, its pros and cons can be found in relative handbooks. Otherwise, it may be necessary to make out the pros and cons via specific analysis.

(2) Solution Instantiating

The solutions discovered initially are generally described as plain text. They need to be instantiated for the purpose of subsequent synthesizing. Instantiated solutions are concrete design fragments modeled with architecture entities, their attributes, and their connections. These design fragments reflect the semantics of informally described solutions. We use the following elements to model solution instances:

- *Components*, represent the computation elements of a solution. Components have their names and attributes.

- *Connectors*, represent the interaction protocols between components. Connectors also have their names and attributes.

- *Structures*, represent the topological connections between components and connectors.

For example, the solution "Using a B/S structure" can be instantiated with a Web server component (S), a Browser component (C), and a HTTP connector (N). This solution instance is illustrated in Figure 3 (a). Another example of solution instance "Using security communication" is illustrated in Figure 3 (b).

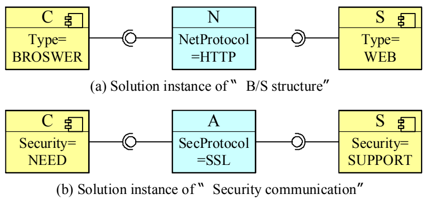
C
Type=
BROSWER
N
NetProtocol
=HTTP
S
Type=
WEB
(a) Solution instance of " B/S structure"
C
Security=
NEED
A
SecProtocol
=SSL
S
Security=
SUPPORT
(b) Solution instance of " Security communication"

Figure 3. Two examples of instantiated solutions

During the instancing process, the components and connectors of all issue solutions need to be considered in the same name space. This is a prerequisite to identify the relations between different issue solutions and combine them in the synthesizing activity.

4.2 Solution synthesizing

Our approach implements the automated synthesis of candidate architecture solutions via three successive steps, based on the issue solutions that have been discovered and instantiated.

(1) Relation identifying

The following types of relations between any two solutions to different issues can be identified:

- *INCLUSIVE*, denoted as *INC*(*ISa*, *ISb*), means that one solution *ISa* includes another solution *ISb*, or *ISb* is part of *ISa*. The effect of *INC*(*ISa*, *ISb*) is that if *ISa* is selected to synthesize architecture solutions, then *ISb* must be selected too. A special case of *INCLUSIVE* relation is *IDENTICAL* relation, which means *ISa* and *ISb* include each other.

The rule to identify *INCLUSIVE* relation is that all components and connectors of *ISb* are included in solution *ISa*, with the same attributes and structures.

- *GENERALIZED*, denoted as *GEN*(*ISa*, *ISb*), means that one solution *ISb* is the generalization of another solution *ISa*, or *ISa* is the specialization of *ISb*. The effect of *GEN*(*ISa*, *ISb*) is that if *ISa* is selected to synthesize architecture solutions, then *ISb* must be selected too.

The rule to identify *GENERALIZED* relation is that *ISa* and *ISb* have the same components, connectors, and structures, and one or more components or connectors of *ISb* are the generalization of corresponding elements of *ISa*.

- *CONFLICTIVE*, denoted as *CON*(*ISa*, *ISb*), means that two solutions *ISa* and *ISb* have conflictive attributes or structures. The effect of *CON*(*ISa*, *ISb*) is that solution *ISa* and *ISb* cannot be selected together to synthesize architecture solutions.

The rule to identify *CONFLICTIVE* attributes is that the same component or connector in *ISa* and *ISb* has different attribute values. The rule to identify *CONFLICTIVE* structure is that the same components and connectors in *ISa* and *ISb* have different topological connections.

- *INDEPENDENT*. If the relation between two solutions is not *INCLUSIVE*, *GENERALIZED*, or *CONFLICTIVE*, they are *INDEPENDENT*. It is freely to select or not select the *INDEPENDENT* solutions together to synthesize architecture solutions.

(2) Solution combining

According to the identified relations between every two solutions to different issues, all feasible combinations of issue solutions can be explored. This step builds a tree representing these combinations. Some branches of the tree must be grown because of the *INCLUSIVE* or *GENERALIZED* relation between issue solutions. Some branches must be pruned because of the *CONFLICTIVE* relation between issue solutions. Figure 4 shows a recursive algorithm for solution combining. To build a whole combination tree, just call *SolutionCombine*(I_0, *SS*=*EMPTY*).

```
SolutionCombine(Ix, SS){
/* Ix:  the issue numbered x.
   SS: a set containing a feasible combination of
        the solutions to issues $I_1$~Ix.
*/
   if ( Ix is the last issue ) return;

   for ( every solution ISb to issue Ix+1 )
      for ( every solution ISa in SS )
         if ( INC(ISa, ISb) || GEN(ISa, ISb) ){
            SolutionCombine(Ix+1, SS+=ISb);
            return;
         }

   for ( every solution ISb to issue Ix+1 )
      for ( every solution ISa in SS )
         if( !CON(ISa, ISb) )
            SolutionCombine(Ix+1, SS+=ISb);
}
```

Figure 4. The algorithm of solution combining

(3) Solution merging

For each feasible combination, the issue solutions are merged to form a candidate architecture solution. The merged architecture solution includes solutions to every issue, so it addresses all of the issues.

As an example, the merging of two issue solutions in Figure 3 is an architecture solution that has a B/S structure and implements a security protocol between the client and the server. Figure 6 shows the merged solution.

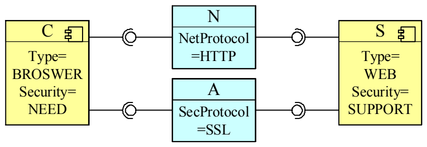

Figure 5. An example of merged solution

5. The capture of design rationale

5.1 Architecture decisions and issue decisions

As aforementioned, the ultimate architecture decision is made by people in the architecture design process. Once this decision has been made, the issue decisions are also settled accordingly. These issue decisions are not explicit but they can be determined by the architecture decision. This relationship is shown by the meta-model in Figure 1.

In our approach, every candidate architecture solution is the synthesis of certain issue solutions. Therefore, if one issue solution is the participant to synthesize the architecture solution that is decided to adopt, then this issue solution is certainly adopted. Otherwise, this issue solution is discarded. This determination relationship can be formulated as below:

$$Decision(ISx) = \begin{cases} ADOPT, & \text{if } ISx \in ASa \\ DISCARD, & \text{otherwise} \end{cases}$$

where *Decision*(*ISx*) denotes the decision on an issue solution *ISx*, *ASa* is the architecture solution adopted, $ISx \in ASa$ means that *ISx* participates to synthesize *ASa*.

5.2 Architecture rationale and issue rationale

The architecture rationale is mainly constituted by two parts. First, based on the synthesis of architecture solutions from issue solutions, the pros and cons of each architecture solution is the summed pros and cons of the issue solutions that participate to synthesize this architecture solution. Second, the rationale of the architecture decision can be supplemented by people, according to their additional architecture evaluation and any other considerations to make the ultimate architecture decision.

The issue rationale is mainly constituted by two parts too. The first part is the pros and cons of this issue solution itself, which has been specified in the step of solution discovering. The second part is based on the architecture rationale. When architecture rationale has been made certain, the rationale of each issue solution can be deduced from the architecture rationale. This relationship is shown by the meta-model in Figure 1.

For the architecture solution adopted, its rationale can be used to explain the reason for adopting the issue solutions that participate to synthesize this architecture solution. Therefore this rationale is also part of the rationale of the issue solution. Similarly, for the architecture solutions discarded, its rationale can also be part of rationale of the issue solutions that participate to synthesize these architecture solutions discarded.

6. Case study

Commanding Display Systems (CDS) are important sub-systems in many large-scale mission-critical applications. A CDS presents rich information onto the desktops of commanders to support mission monitoring, analyzing, and commanding. A CDS usually has a large number of monitors to display the data that is generated by the back-end hosts in real time. A great deal of history data also need to be stored and queried on demand.

This case study is based on a real-life CDS that has served for many years and is currently facing re-architecting. According to the legacy system and new technologies, several key concerns of design are promoted and disputed, e.g., "Employing a CDS server or not?", "Using a general purpose database or a real-time database?" etc. There seems to be no direct way to reach the rational decision on all these options, because they are interdependent and the exploration of their combination is a hard work. This case study illustrates how the approach proposed in this paper can provide an assistant to overcome the obstacles and achieve the target architecture solution.

We firstly list the primary functional requirements (FRs) and non-functional requirements (NFRs) of this CDS, as shown in Table 1.

Table 1. FRs and NFRs of the CDS

Type	Requirement	Description
FR	*R*1	Live data display
	*R*2	History data display
	*R*3	Data persistence
NFR	*R*4	Real-time
	*R*5	Non-overload
	*R*6	Visualization
	*R*7	Maintainability

According to the FRs and the legacy systems, three kinds of basic components can be determined to form the original architecture. They are a set of Monitors (Ms), a Data Source (DS), and a Database (DB). Their attributes and connections are to be resolved.

Based on the FRs, NFRs, and the original architecture, we can elicit architecturally significant issues. Table 2 lists the elicited issues and the requirements which they address.

Table 2. Elicited issues of the CDS

Issue	Description	Requirements
*I*1	Real-time live data acquiring	*R*1, *R*4
*I*2	Quickly history data acquiring	*R*2, *R*4
*I*3	DB load	*R*3, *R*5
*I*4	Visualization	*R*6
*I*5	Maintainability	*R*7

We then discover the solutions to each elicited issue. Table 3 shows these solutions in brief, including the description, pros, and cons of each solution.

Table 3. Discovered issue solutions of the CDS

Issue	Issue Solution	Description, Pros, and Cons of each solution
*I*1	*IS*1.1	**Desc**: DS pushes data to Ms directly; **Pros**: Simple to implement; normal DB technology; **Cons**: Lack of management for live data.
	*IS*1.2	**Desc**: DS puts data into a real-time DB; Ms get data from that DB; **Pros**: Special support by real-time DB; live data and history data can be managed; **Cons**: Need a real-time DB; increase the cost of development.
*I*2	*IS*2.1	**Desc**: Cache the data in Ms; **Pros**: Simplify other parts of the system; **Cons**: Ms may be too complex and need large resources.
	*IS*2.2	**Desc**: Use a cache server; **Pros**: Ms are simple; **Cons**: Need a server to support cache.
	*IS*2.3	**Desc**: Use a real-time DB; **Pros** and **Cons**: (as *IS*1.2)
*I*3	*IS*3.1	**Desc**: Cache the data in Ms; **Pros** and **Cons**: (as *IS*2.1)
	*IS*3.2	**Desc**: Use a cache server; **Pros** and **Cons**: (as *IS*2.2)
*I*4	*IS*4.1	**Desc**: Graphical application; **Pros**: High graphical performance; **Cons**: Difficult to maintain.
	*IS*4.2	**Desc**: Rich Internet Application (RIA); **Pros**: Easy to maintain; **Cons**: Graphical performance is not very high.
*I*5	*IS*5.1	**Desc**: Auto-update client; **Pros**: Support complex application; **Cons**: Need a server to support update.
	*IS*5.2	**Desc**: Browser/Server (B/S); **Pros**: High maintainability; **Cons**: Limited graphical performance.

Figures 6-10 show the results of solution instantiating for each issue solution.

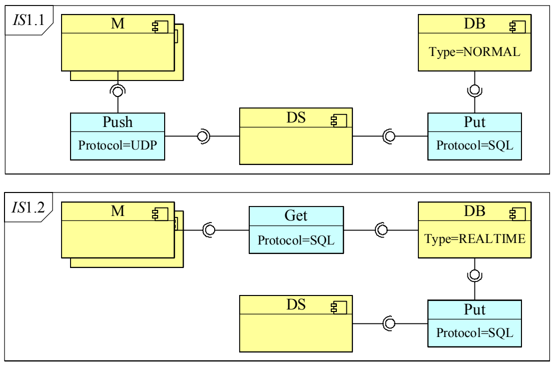

Figure 6. Instantiated solutions to issue ***I*1**

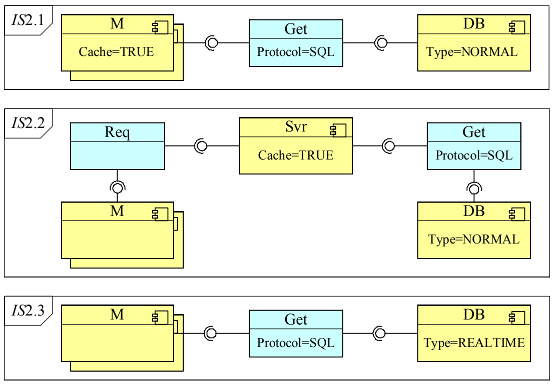

Figure 7. Instantiated solutions to issue ***I*2**

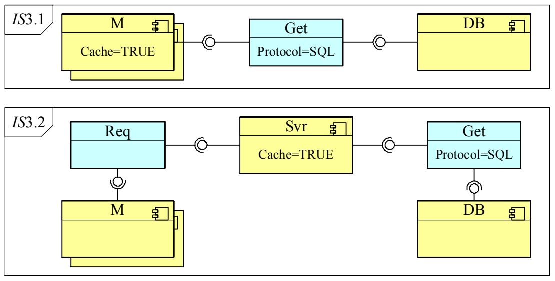

Figure 8. Instantiated solutions to issue *I3*

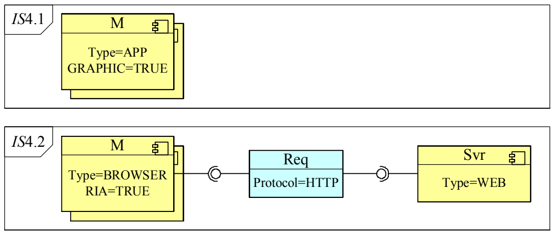

Figure 9. Instantiated solutions to issue *I4*

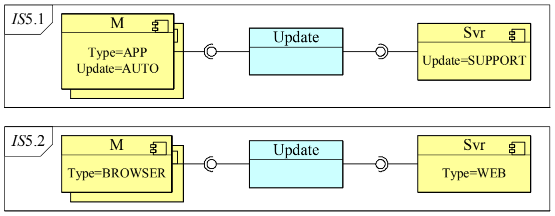

Figure 10. Instantiated solutions to issue *I5*

The next activity is automated solution synthesis. Firstly, one *INCLUSIVE*, three *GENERALIZED*, and five *CONFLICTIVE* relations between the issue solutions are identified. For example, *IS*2.3 is part of *IS*1.2, so *INC*(*IS*1.2, *IS*2.3) is true; the DB in *IS*2.1 is a specialization of the DB in *IS*3.1, so *GEN*(*IS*2.1, *IS*3.1) is true; the DB in *IS*1.1 has different attribute value from the DB in *IS*2.3, so *CON*(*IS*1.1, *IS*2.3) is true. Table 4 shows these relations.

Table 4. Relations between issue solutions

Relation	Issue Solutions
INC	(*IS*1.2, *IS*2.3)
GEN	(*IS*2.1, *IS*3.1), (*IS*2.2, *IS*3.2), (*IS*4.2, *IS*5.2)
CON	(*IS*1.1, *IS*2.3), (*IS*1.2, *IS*2.1), (*IS*1.2, *IS*2.2), (*IS*4.1, *IS*5.2), (*IS*4.2, *IS*5.1)

Based on the identified relations, issue solutions can be combined. Figure 11 shows the combination tree of all issue solutions. Some branches of the tree are pruned because of certain relations between the issue solutions. For example, the branch between *IS*2.1 and *IS*3.2 is pruned because of *GEN*(*IS*2.1, *IS*3.1), meaning that if *IS*2.1 is included in the combination then *IS*3.1 must be included, instead of *IS*3.2; the branch between *IS*4.1 and *IS*5.2 is pruned because of *CON*(*IS*4.1, *IS*5.2), meaning that *IS*4.1 and *IS*5.2 cannot be included together. The tree shows that there are totally eight feasible combinations.

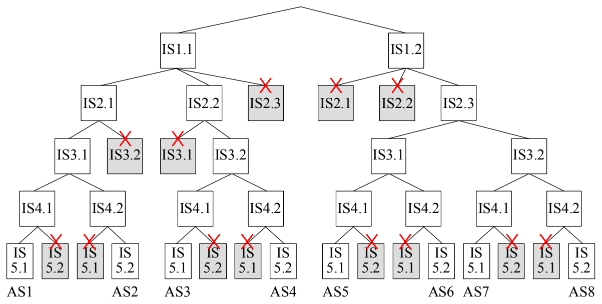

Figure 11. Combination tree of issue solutions

The last step of solution synthesis is to merge the issue solutions of each feasible combination. The results are eight candidate architecture solutions, named *AS*1~*AS*8. Figures 12 and 13 show two of them: *AS*1 and *AS*8.

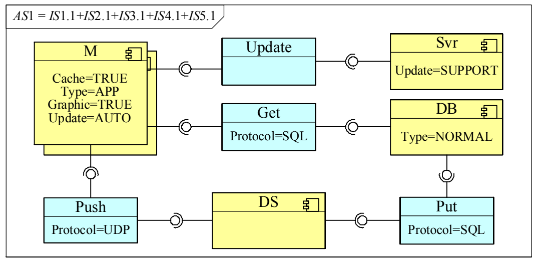

Figure 12. Synthesized architecture solution ***AS*1**

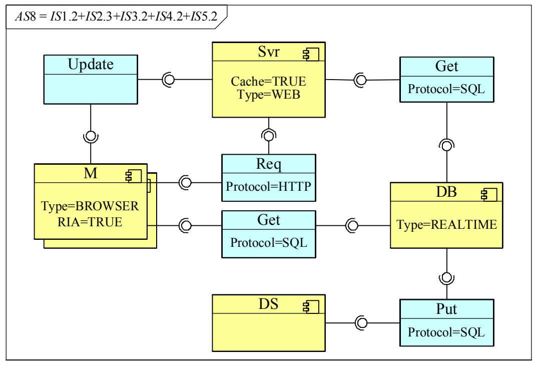

Figure 13. Synthesized architecture solution ***AS*8**

For example, architecture solution *AS*8 is synthesized from the issue solutions *IS*1.2, *IS*2.3, *IS*3.2, *IS*4.2, and *IS*5.2. This architecture solution means: a real-time DB is used to store all data generated by the DS; the Ms get live data from the DB directly; the Ms use a RIA mode to get history data from the DB via a Web server; the Web server also acts as a cache server. The pros and cons of this architecture solution can be accumulated from those of the corresponding issue solutions.

From the synthesized architecture solutions, architects and customers make their choice, based on the summed pros and cons, and other additional evaluations of each architecture solution. When the architecture solution has been selected, the decisions and rationale of each issue can be deduced.

For example, suppose that the ultimate architecture decision is to adopt *AS*8. The rationale of *AS*8 is accumulated pros of *IS*1.2, *IS*2.3, *IS*3.2, *IS*4.2, *IS*5.2, as well as other considerations specified by architects. From this architecture decision and rationale, the decisions on *IS*1.2, *IS*2.3, *IS*3.2, *IS*4.2, and *IS*5.2 are *ADOPT*, and the rationale of these issue solutions are their own pros and the rationale of *AS*8. Tables 5 and 6 show all captured decisions and rationale of architecture solutions and issue solutions.

Table 5. Architecture decisions and rationale

Arch Solution	Arch Decision	Arch Rationale
*AS*1	*DISCARD*	*Cons*(*IS*1.1, *IS*2.1, *IS*3.1, *IS*4.1, *IS*5.1), other considerations
*AS*2	*DISCARD*	*Cons*(*IS*1.1, *IS*2.1, *IS*3.1, *IS*4.2, *IS*5.2), other considerations
*AS*3	*DISCARD*	*Cons*(*IS*1.1, *IS*2.2, *IS*3.2, *IS*4.1, *IS*5.1), other considerations
*AS*4	*DISCARD*	*Cons*(*IS*1.1, *IS*2.2, *IS*3.2, *IS*4.2, *IS*5.2), other considerations
*AS*5	*DISCARD*	*Cons*(*IS*1.2, *IS*2.3, *IS*3.1, *IS*4.1, *IS*5.1), other considerations
*AS*6	*DISCARD*	*Cons*(*IS*1.2, *IS*2.3, *IS*3.1, *IS*4.2, *IS*5.2), other considerations
*AS*7	*DISCARD*	*Cons*(*IS*1.2, *IS*2.3, *IS*3.2, *IS*4.1, *IS*5.1), other considerations
*AS*8	*ADOPT*	*Pros*(*IS*1.2, *IS*2.3, *IS*3.2, *IS*4.2, *IS*5.2), other considerations

Table 6. Issue decisions and rationale

Issue Solution	Issue Decision	Issue Rationale
*IS*1.1	*DISCARD*	*Cons*(*IS*1.1), *Rationale*(*AS*1, *AS*2, *AS*3, *AS*4)
*IS*1.2	*ADOPT*	*Pros*(*IS*1.2), *Rationale*(*AS*8)
*IS*2.1	*DISCARD*	*Cons*(*IS*2.1), *Rationale*(*AS*1, *AS*2)
*IS*2.2	*DISCARD*	*Cons*(*IS*2.2), *Rationale*(*AS*3, *AS*4)
*IS*2.3	*ADOPT*	*Pros*(*IS*2.3), *Rationale*(*AS*8)
*IS*3.1	*DISCARD*	*Cons*(*IS*3.1), *Rationale*(*AS*1, *AS*2, *AS*5, *AS*6)
*IS*3.2	*ADOPT*	*Pros*(*IS*3.2), *Rationale*(*AS*8)
*IS*4.1	*DISCARD*	*Cons*(*IS*4.1), *Rationale*(*AS*1, *AS*3, *AS*5, *AS*7)
*IS*4.2	*ADOPT*	*Pros*(*IS*4.2), *Rationale*(*AS*8)
*IS*5.1	*DISCARD*	*Cons*(*IS*5.1), *Rationale*(*AS*1, *AS*3, *AS*5, *AS*7)
*IS*5.2	*ADOPT*	*Pros*(*IS*5.2), *Rationale*(*AS*8)

In the real project, the ultimate architecture adopted is *AS*8, with some slight refinements. The automated solution synthesis and rationale capture provided by this approach make the exploration of solution space and the comprehension of architecture decision a systematic and rational process.

7. Discussion

This approach does not intend to replace people's work totally in the architecture design. Instead, it provides automated support in the key activities to alleviate the difficulties of architecting. Although the exploiting of issue solutions in this approach is accomplished manually by now, we think that the complexity of design has been reduced because people has been released from the difficulty of architecture solution derivation. The automated solution synthesis lets people concentrate on the solutions to relatively simple and specific issues, and then get the candidate architecture solutions easily. The benefits of "Divide and Conquer" are realized in this way. Furthermore, the unfeasible combinations of issue solutions are eliminated by the automated synthesis, so that the number of generated candidate architecture solutions will be much less than the full combination of issue solutions. For example, in the preceding case study, the number of full combination of issue solutions is 2*3*2*2*2=48, whereas the actual number of synthesized architecture solutions is eight. Thus the work of comparison and selection of these solutions is greatly simplified. On the other hand, although people make ultimate architecture decisions and supplement rationale for their decisions, the corresponding decisions and rationale of issue solutions can be deduced automatically, based on the synthesis relationship between architecture and issue solutions. This deduced information is useful for reasoning, comprehension, and communication of the architecture artifacts. The burden of manually tracking and recording this knowledge is eased.

This approach is not a whole solution to architecting either. Instead, this approach can accommodate many existing methods for architecture design. First, although we have not given support to accomplish issue eliciting yet, there are successful methods, e.g., Siemens S4V [6], that provide supports for this task, and Hofmeister et al. [1]'s general model of architecture design has included this activity, i.e., architectural analysis. Second, although our approach has not yet provided automated support for discovering issue solutions, there are also rich set of methods and knowledge can be leveraged here, i.e., various design patterns, architecture patterns, etc. In fact, because we concern separated issues instead of the whole architecture, those patterns can be applied more directly and effectively. Third, this approach does not address the task of architecture evaluation, which is included in the architecture deciding activity. There have been a number of proposed architecture evaluation [18] and decision-making methods [8, 9] . These methods also account for the reasonability and applicability of our architecture design process.

Last but not least, we point out that this approach is applicable because it works at the level of architecture. As the highest level design, it should focus on the architecturally significant issues, which need to be limited within a relatively small amount. According to the famous "7 ± 2" principle [19], we think that about nine is a reasonable upper bound. Within this scope, the number of issue solutions and the synthesized candidate architecture solutions can be under the human intellectual control.

8. Conclusions and future work

We have proposed a decision-centric architecture design method that implements automated architecture solution synthesis and design rationale capture. This approach can provide pragmatic help to architects, in the process from architecturally significant issues to candidate architecture solutions. This approach also integrates the notions of decisions and rationale into the architecture design process, and reduces the overhead of rationale capturing and recording, so that the process can be more efficient and productive.

This paper presents our initial work towards practical architecture design method and tool support. We are now working on the implementation and integration of this approach with our former architecture description language, ABC/ADL [20], and the corresponding architecture tool . We also plan to add behavior concerns to enhance the modeling capability of issue solutions. Other future work includes the further validation and elaboration of this approach in real-life applications.

9. Acknowledgements

This effort is sponsored by the National Basic Research Program of China (973) under Grant No. 2005CB321805 and the National Natural Science Foundation of China under Grant No. 90412011, 90612011, 60403030, 60503028.

10. References

[1] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A General Model of Software Architecture Design Derived from Five Industrial Approaches," *The Journal of Systems and Software*, vol. 80, no. 1, Jan. 2007, pp. 106-126.

[2] J. Bosch, "Software Architecture: the Next Step," *Proc. 1st European Workshop on Software Architecture (EWSA'04)*, Springer-Verlag 2004, pp. 194-199.

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

[4] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.

[5] H. d. Bruin and H. v. Vliet, "Quality-Driven Software Architecture Composition," *The Journal of Systems and Software* vol. 66, no. 3, June 2003, pp. 269-284

[6] C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*. Addison-Wesley, 2000.

[7] P. Kruchten, *The Rational Unified Process: An Introduction*, 3rd ed. Addison-Wesley, 2003.

[8] M. Svahnberg, C. Wohlin, L. Lundberg, and M. Mattsson, "A Quality-Driven Decision-Support Method for Identifying Software Architecture Candidates," *Int'l Journal of Software Eng. & Knowledge Eng.*, vol. 13, no. 5, Oct. 2003, pp. 547-573.

[9] T. Al-Naeem, I. Gorton, M. A. Babar1, F. Rabhi, and B. Benatallah, "A Quality-Driven Systematic Approach for Architecting Distributed Software Applications," *Proc. 27th Int'l Conf. Software Eng. (ICSE'05)* 2005.

[10] D. E. Perry and A. L. Wolf, "Foundations for the Study of Software Architecture," *ACM SIGSOFT Software Eng. Notes* 1992, pp. 40-52.

[11] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," *IEEE Software* vol. 22, no. 2 2005, pp. 19-27.

[12] P. Kruchten, "An Ontology of architectural design decisions in software-intensive systems," *Proc. 2nd Groningen Workshop on Software Variability Management*, Rijksuniversiteit Groningen, Dec. 3-4 2004, pp. 54-61.

[13] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," *Proc. 5th Working IEEE/IFIP Conf. on Software Architecture (WICSA'05)* 2005.

[14] A. Jansen, J. v. d. Ven, P. Avgeriou, and D. K. Hammer, "Tool Support for Architectural Decisions," *Proc. 6th Working IEEE/IFIP Conf. on Software Architecture (WICSA'07)* 2007.

[15] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," *The Journal of Systems and Software*, vol. 80, no. 6, June 2007, pp. 918-934.

[16] A. Akerman and J. Tyree, "Using Ontology to Support Development of Software Architectures," *IBM Systems Journal*, vol. 45, no. 4 2006, pp. 813-825.

[17] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, 1996.

[18] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architecture*. Addison-Wesley, 2002.

[19] G. Miller, "The Magic Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *The Psychological Review*, vol. 63 1956, pp. 81-97.

[20] H. Mei, F. Chen, Q. Wang, and Y. D. Feng, "ABC/ADL: An ADL Supporting Component Composition," *Proc. 4th Intl. Conf. on Formal Eng. Methods: Formal Methods and Software Eng. (ICFEM'02), LNCS 2459*, Springer-Verlag, 2002, pp. 38-47.