An Integrated Approach to Quality Achievement with Architectural Design Decisions

Heeseok Choi
Korea Institute of Science and Technology Information, Daejeon, Republic of Korea
Email: choihs@kisti.re.kr

Youhee Choi
Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea
Email: yhchoi@etri.re.kr

Keunhyuk Yeom
Pusan National University, Busan, Republic of Korea
Email: yeom@pusan.ac.kr

***Abstract*—Good software architecture is critically important for successful software development. The software architecture can be defined as a set of architectural design decisions. However, the effort for acquiring good software architecture currently lacks of interest and experience in architectural design decisions. For this reason, during acquiring good software architecture, it is difficult to evaluate the architectural designs and make changes to the architecture that is relevant for the required changes. Therefore, this paper proposes a model of architectural design decisions for making architectural design decisions more explicit. Based on the proposed model, this paper also proposes an integrated approach for acquiring good software architecture with respect to its requirements, which is called AQUA. Namely, the AQUA defines decision-centric process of finding, evaluating, and changing the decisions. During the decision-centric process, the AQUA involves works of architectural evaluation and transformation. The AQUA provides software architects with a means for achieving software quality attributes through acquiring good software.**

***Index Terms*—architectural design decision, software architecture, architectural evaluation, architectural transformation, architectural change**

I. Introduction

Software architecture has become a very important concept in research as well as industry. Moreover, good software architecture is critically important for successful software development. The software architecture can be defined as a set of architectural design decisions[1,2]. Therefore, architectural design decisions are closely related to software development process.

Although the importance of making good design decisions was emphasized, e.g. during design, evaluation, and transformation of software architecture[3], the design decisions themselves are not explicitly handled, and are not closely related to the development process of software architecture. Namely, the effort for acquiring good software architecture currently lacks of interest and experience in architectural design decisions. Consequently, the results of the design decisions underlying the architecture are implicitly embedded within the architecture. For this reason, during acquiring good software architecture, it is difficult to evaluate the architectural designs and make changes to the architecture that is relevant for the required changes.

This paper proposes a model of architectural design decision for making architectural design decisions more explicit. Based on the proposed model, this paper also proposes an integrated approach for acquiring good software architecture with respect to its requirements, which is called AQUA. During performing the proposed approach, it is important to understand relationships between requirements and an architecture. Architectural design decisions play a role of bridge between requirements and an architecture as a fundamental concept that realizes one or more requirements on a given architecture. Since the design decisions represent solutions to design problems, it makes sense to use them to evaluate how well the solution satisfies its purpose with respect to software requirements. According to the results, architectural changes such as architectural additions, subtractions, and modifications to the software architecture may be required. Namely, it may occur to reconstruct the design decisions underlying the current architecture that are relevant for the required changes. It is important to make architectural changes in harmony with the existing design decisions.

The remainder of this paper is organized as follows. In section 2, the problems of software architecture with respect to architectural design decisions are explained in more detail. The proposed model of architectural design decisions is presented in section 3. The next section presents an integrated approach for finding, analyzing, and changing architectural design decisions, which is

called AQUA. The AQUA is applied to an example in section 5. Then the AQUA is compared with related work in section 6. The paper concludes with future work and conclusions in section 7.

II. Problems of Software Architecture

Software architecture is the result of the architectural design decisions made over time. Nevertheless, the effort for acquiring good software architecture currently lacks of interest and experience in architectural design decisions. In other words, architecture-based development process is focused on the resulting artifacts, instead of the decisions that lead to them. Although the effects of the made decisions are present in the design, the decisions themselves are not explicitly handled, and are not closely related to the development process of software architecture. Clearly, architectural design decisions currently lack a representation in design and use of software architecture. This leads to a number of problems associated with software architecture:

- Design decisions are often intertwined with each other[1], as they work in close relationship together. Furthermore, they typically affect multiple parts of the design simultaneously. This leads to the situation that the design decision information is fragmented across various parts of the design, making it hard to find and change the decisions.
- Software architecture realizes one or more requirements on a given architecture. Nevertheless, the relationships between requirements and architecture are ambiguous. This makes it difficult to understand and control quality attributes in the architecture.
- Making a change to the architecture still depends on architects' experiences or intuition. During architectural change for acquiring good software architecture, it is difficult to reconstruct the design decisions.

III. Architectural Design Decisions in the Life Cycle

Architectural design decisions play an important role in the design and use of software architectures[3]. Namely, architectural design decisions are made and evolved through architecture-based development life cycle. In design, it is important to provide architectural designs incorporating architectural design decisions. In evaluation, architectural design decisions are closely related to revealing any potential defects or assessing the fulfillment of required quality requirements. In transformation, architectural design decisions play a significant role in reducing defects in the architecture or making changes to the architecture. It is important to do this in harmony with the existing design decisions. Consequently, the architectural design decision is the central concept in the development and evolution of software architectures[4,5,6]. In order to make architectural design decisions more explicit in the life cycle, we developed a model of architectural design decisions. Figure 1 provides an informative summary of the key concepts introduced by this approach and their inter-relationships. Figure 1 presents these concepts in the context of software architecture development and its evolution.

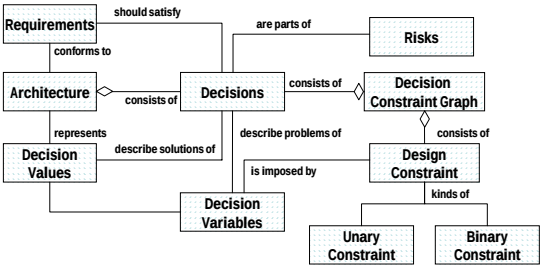

Figure 1. Model of architectural design decisions

- **Software requirements** describe expectations for functionality and quality of software. Software requirements are divided into functional requirements and non-functional requirements. Functional requirements are used to determine interesting designs, and non-functional requirements are used to identify quality attributes. Architectural design decisions about interesting designs should satisfy their software requirements.
- **Software architecture** describes the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution[7]. Every system has an architecture. Since software architecture realizes software requirements, software architecture should conform to its software requirements.
- **Decisions** describe the outcome of a design process. Software architecture consists of a set of decisions. The decisions identify the system's key structural elements, their externally visible properties, their relationships, and design mechanisms. The decisions typically affect multiple parts of designs simultaneously. Examples of such decisions are architectural styles and design patterns. The decision is composed of a pair of decision variable and decision value.
- **Decision value** describes a design itself applied to the current architecture as the selected solution out of design alternatives applicable to each design problem. The decision values can be easily conceived from well presented software architecture.
- **Decision variable** describes the architectural design problem that each selected solution is addressing, such as "What are the big parts of the system?" and/or "How are they connected?". Such decision problem can be found by analyzing decision values based on architectural knowledge such as design patterns, styles, and architectural views.
- **Risks** represent decisions having any potential defects in the architecture. Namely, they are parts of decisions. To solve the described risks, one or more

changes can be proposed. For each of the proposed changes, software architects should make changes to the architecture after performing a change impact analysis based on a decision constraint graph.

- **Design constraints** capture any constraints to the designs that the chosen alternative (the decision) might pose. The decisions introduce two kinds of design constraints. One is unary constraint and the other is binary constraint.
- **Unary constraint** captures any constraint to the design which restricts design alternatives applicable to each design issue. In order to determine unary constraints, software architects should first analyze the characteristics of decision values at various points in the design. For example, if the design elements support the concurrency of system, it can be considered that there is the constraint equal to concurrency support. Next, software architects should determine whether the characteristics are closely related to the requirements specified in previous evaluation contract. Finally, the characteristics irrelevant to requirements should be excluded.
- **Binary constraint** captures any constraint for design consistency that two decision values might pose each other, which represents a condition restricting design alternatives applicable to relevant decision variables. In order to determine binary constraints, software architects should analyze only the characteristics causing consistency problems among decision values.
- **Decision constraint graph** is the vehicle for change impact analysis through providing consistency information among design decisions. Changes in the architecture will have an effect on other decisions. It is therefore important to have a relationship among decisions. Figure 2 represents the generation of a decision constraint graph. Each node in the graph represents a decision variable, and each edge in the graph represents constraint relationship between two decision variables. To generate a decision constraint graph, software architects should first document two kinds of constraints according to the above illustration; unary constraints and binary constraints. Based on the identified constraints, the relationships among design decisions are determined with respect to design consistency. As a result, nodes and edges of decision constraint graph are defined. Reuse of software architecture is the use of earlier tried and tested combinations of design decisions (e.g. design patterns or components).

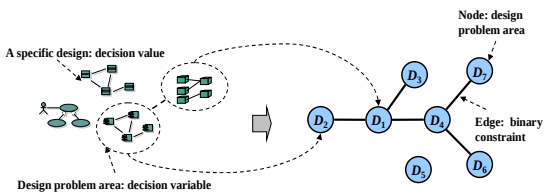

Figure 2. Decision constraint graph

IV. Decision-Centric Approach

We proposed a model of architectural design decisions for making them more explicit in the previous section. Based on the proposed model, we propose an integrated approach for acquiring good software architecture with respect to its requirements, which is called AQUA. The AQUA supports finding, analyzing, and changing decisions based on the model of architectural design decisions. During the process, the AQUA easily supports the integration of architectural evaluation with architectural transformation. Therefore, the AQUA provides software architects with a means for achieving quality attributes through acquiring good software architecture.

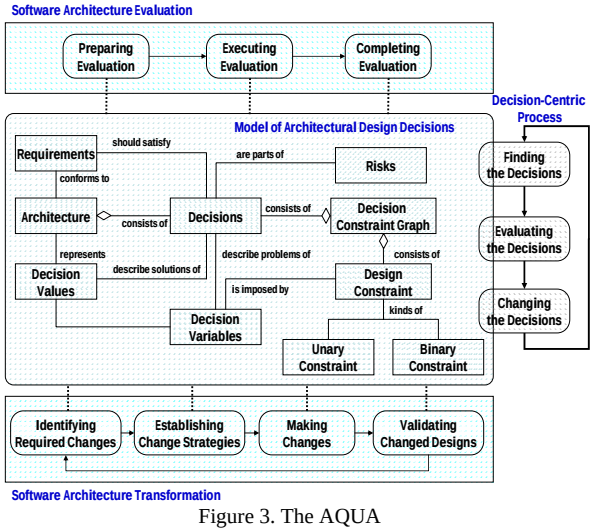

Figure 3. The AQUA

Figure 3 presents an overview of the AQUA. As shown in Figure 3, the AQUA presented the model of architectural design decisions for making architectural design decisions more explicit. Based on the proposed model, the AQUA defined decision-centric process of finding, evaluating, and changing the decisions. During the decision-centric process, the AQUA involved works of architectural evaluation and transformation. Namely, architectural evaluation is closely related to finding and evaluating decisions, and architectural transformation is related to changing them for acquiring good software architecture. The works of finding, analyzing, and changing decisions are discussed in the below.

Finding the Decisions: Software architecture is composed of architectural design decisions, which represent the aspects of an architecture that have a significant impact on achieving quality attributes. In order to find architectural design decisions, it is important to characterize quality attributes influencing on design decisions. In addition, it is necessary to present software architecture helpful for finding decisions. Through performing the works, architectural design decisions can be explicitly found as pairs of decision variables and decision values.

Evaluating the Decisions: Once architectural design decisions have been found, decision-centric evaluation can be performed with respect to software requirements.

Some design decisions may prevent satisfying quality requirements due to conflicts among quality goals or mismatch problems among decisions. Therefore, we can understand whether the architecture is good or not. In addition, designs not to satisfy quality requirements can be determined through architecture evaluation.

Changing the Decisions: Software architects can apply various changes in order to reduce any potential defects, which may be conflict decisions or unsatisfactory decisions with respect to quality requirements. Each change leads to a new version of the architecture that has the same functionality, but different satisfaction for desired quality attributes. However, before applying architectural changes to the architecture, it is necessary to perform a change impact analysis due to relationships among decisions. Based on the results, good software architecture can be acquired through reconstruct designs that are relevant for the required changes.

A. Software Architecture Evaluation

In the AQUA, software architecture evaluation plays a significant role in finding and evaluating architectural design decisions. To do this, it is divided into three main areas of activities: preparing the evaluation, executing the evaluation, and completing the evaluation. During these activities, architectural design decisions that have profound impacts on the achievement of quality attributes are systematically found, reasoned through, and analyzed. Also, the defects in a design are reported based on the decisions. Figure 4 presents the process of software architecture evaluation in the AQUA.

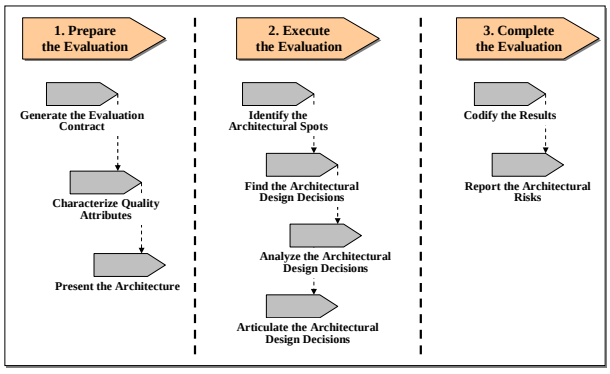

Figure 4. The process of software architecture evaluation

Phase 1. Prepare the Evaluation: In this phase, it is important to explicitly characterize quality attributes by their quality requirements. Also, it is necessary to properly present the architecture for finding architectural design decisions. In this phase, the following steps are performed. To do this, the following steps are performed.

- *Generate the evaluation contract*. Generating an evaluation contract determines how many requirements will be handled and the kinds of quality attributes that will be dealt with. Its result clarifies the scope and goals of an architectural evaluation. During this step, expectations from the architectural evaluation can be concluded and negotiated.
- *Characterize quality attributes*. Since architectural evaluation focuses on quality attributes, it is important to have clear and informative characteristics for each quality attribute. In this step, it is necessary to use the wealth of knowledge that already exists in the various quality attribute research groups.
- *Present the architecture*. It is noted that the large variance of a quality assessment based on architectural analysis is associated with the granularity of the system description necessary to perform an evaluation. In this respect, the 4+1 view model of architecture[8] in UML is one solution to obtain good architectural documentation. Through well presenting the architecture, architectural design decisions can be more explicitly found.

Phase 2. Execute the Evaluation: During this phase, software architects find architectural design decisions and analyze them with respect to quality attributes. In this phase, the following steps are performed.

- *Identify the architectural spots*. Once the functional requirements have been defined and an architecture has been presented, we can find architectural spots associated with the functional requirements of the evaluation contract. Architectural spots represent the areas of significant architectural designs with respect to the evaluation. The result from these steps is the architectural spots on which the evaluation is focused.
- *Find the architectural design decisions*. Since architectural design decisions have a profound impact on the achievement of quality attributes, it is very important to find and analyze them during architectural evaluation. Software architects should summarize key designs from the architectural spots. Then decision values can be identified through characterizing design summaries. Finally, decision variables are determined by identifying one or more design issues that each decision value reveals.
- *Analyze the architectural design decisions*. Once the architectural design decisions have been found, they must be separately analyzed. Analysis in this step does not entail detailed simulation or precise mathematical modeling. It is more of a qualitative analysis for revealing defects in a design. When we reason through the decisions, we may use the various design theories[9,10,11] or refer to other alternatives from competing architectures.
- *Articulate the architectural design decisions*. Architectural design decisions affect one or more system qualities (performance, availability, modifiability, security, and so on). In addition, the decisions involve trade-offs because architects make decisions in complex circumstances. Properly documenting the decisions' impacts on the qualities is useful for providing insights concerned with the quality achievement of architecture with respect to

its desired qualities. It helps in determining the tradeoffs among quality attributes and understanding the effects made by changing architectural design.

Phase 3. Complete the Evaluation: During this phase, the previous results are incorporated in a way presenting relationships between requirements and an architecture centering on architectural design decisions. It allows us to understand whether an software architecture is satisfactory with respect to its requirements or not. In this phase, the following steps are performed.

- *Codify the results*. Systematically codifying the relationships between architecture and quality attributes greatly enhances the ability of understanding and controlling quality attributes in the architecture.
- *Report the architectural risks*. Architectural risks represent decisions having any potential defects in the architecture. Therefore, they provide hints as to what parts should be changed afterwards.

B. Software Architecture Transformation

In the AQUA, software architecture transformation plays a role in changing architectural design decisions. It is divided into four phases of activities: identifying transformation requests, establishing transformation strategy, transforming architecture, and verifying transformation results. During these activities, the decisions requiring changes are identified, and relationships among decisions are understood. Then it is performed to reconstruct designs that are relevant for the required changes, and to validate the results. Figure 5 presents the process of software architecture transformation in the AQUA.

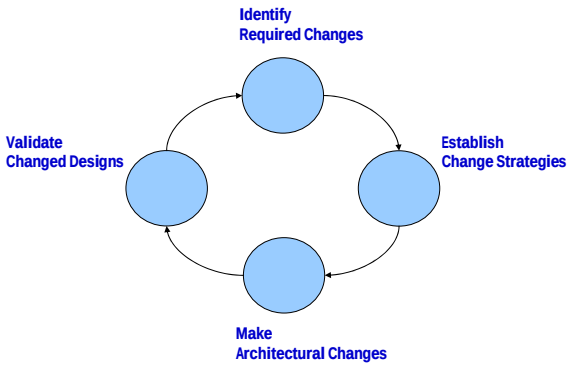

Figure 5. The process of software architecture transformation

Phase 1. Identify Required Changes: The goal of this phase is to determine the decisions requiring changes, which represent the decisions having any potential defects. To do this, software architecture evaluation is performed according to the method illustrated in the previous section. Namely, this phase is closely related to software architecture evaluation. In order to acquire good software architecture, conflict decisions or unsatisfactory decisions with respect to software requirements should be reduced.

.

Phase 2. Establish Change Strategies: The goal of this phase is to establish strategies for architectural changes. The strategies are represented as a series of changes by a change impact analysis. In other words, making a change to a specific decision affects relevant decisions. Therefore, the change to the specific decision may accompany a series of changes. Therefore, it is necessary to analyze how a change to the specific decision affected on other decisions, and to determine an applicable alternative from various ones in order to establish change strategies. The detailed steps in the phase are performed as follows.

- *Find architectural alternatives*. Architectural alternatives are alternative solutions to each design problem. To find architectural alternatives, we can utilize various design theories or refer to other alternatives from candidate architectures or architect's experience.
- *Analyze architectural alternatives*. To determine applicable architectural alternatives, software architects should analyze the architectural alternatives in terms of the scope and impacts of applying them. Architectural alternatives should be consistent with other decisions. To achieve this goal, we propose checking consistencies among decisions based on the decision constraint graph. Through using the decision constraint graph, changing strategies representing a series of changes are established as candidate strategies.
- *Determine changing strategies*. A series of design alternatives satisfying both node consistency and arc consistency has to be selected as an architectural strategy among several design alternatives.

Phase 3. Make Architectural Changes: The goal of this phase is to realize architectural changes according to the changing strategies. As a result, a new version of the architecture satisfying one or more of its desired qualities is redefined.

Phase 4. Validate Changed Designs: The goal of this phase is to confirm or validate whether the changed architecture is good with respect to its requirements or not. To do this, architectural mismatch has to be reevaluated including changed design decisions. In addition, quality satisfaction has to be evaluated by analyzing relationships between quality requirements and the architecture. The detailed steps in the phase are performed as follows.

- *Reevaluate conflicts among decisions*. Through using the decision constraint graph, software architects can confirm whether the changed architecture includes decision conflicts or not.
- *Reevaluate quality satisfaction*. Architectural evaluation can be again used in order to determine a software architecture's fitness with respect to its desired quality attributes. The changed software architecture has to be evaluated with respect to its desired quality requirements.

V. AN ILLUSTRATIVE EXAMPLE

To address the practical applicability and features of our approach, we have chosen an example that is rich with interesting designs and architectural problems, that of National science and Technology Information System, which is called NTIS. The NTIS will provide various types of statistical information on experts and researchers to support executive decision makers and other research communities. This helps improve the efficiency of R&D equipment/resource utilization and sharing, and ensure end-to-end life cycle management of R&D projects. The NTIS has the following characteristics:

- **Heterogeneous**, especially data formats and semantics. There is no common schema in the project data across R&D centers. Since R&D centers have their own specific private data, data from R&D centers are heterogeneous.
- **Distributed**, e.g. integration of existing disconnected R&D project information system. Each R&D center has its own system to maintain its project data, and their systems are connected through public network. Therefore, the NTIS should ensure the secure data transportation over public network.
- **Change**, e.g. change of data schema, addition of R&D centers, change of DBMS, evolution of information demands from stakeholders. The NTIS is dynamic because of a large scale of diverse participants (Institutes, R&D centers) and stakeholders (Ministries, Public, and Industry).

Figure 6 presents the architecture of the NTIS. This architecture contains an Enterprise Service Bus (ESB) as its backbone. The elements of the NTIS architecture are the followings:

- **Application Services.** The applications are exposed as services on the ESB. They are reused and interacted under the control and management inside ESB.
- **Business Process Monitoring.** All the business processes and activities are managed inside process management server. The business processes are weaved from services on the ESB. The key business performance indicators are generated and gathered during the execution of business processes, the enterprise can monitor them according to the events they emit.
- **Information Services.** The information services are unified representation of business data that may come from or aggregate different data sources. With this layer, the enterprise makes a consistent view of data and single access point for that data. It will greatly reduce the complexity for specific information retrieving inside the enterprise.
- **Service Registry.** The service registry is used as the management tool to control the services inside the enterprise. With this tool, the enterprise can control the services definition, providing all services' definition inside the enterprise. All the services consumers can query the necessary information for services that they want to use.
- **Portal.** The portal solution helps the componentization of user interaction, which greatly improves the reusability from the user interface level. It also provides the framework for customization of user interface level for different users.

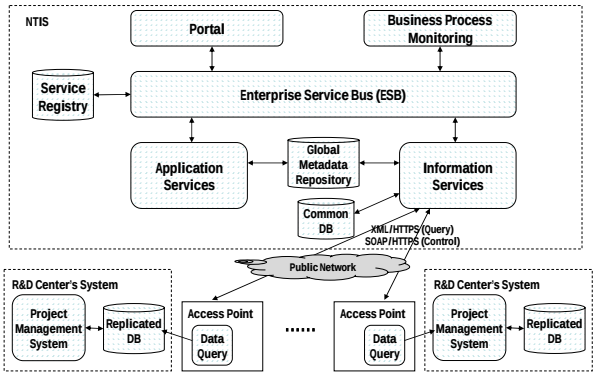

Figure 6. The architecture of the NTIS

A. Finding the Decisions

In this example, we found some interesting decisions as the followings.

- **Decision D01: Development Paradigm.** Service oriented architecture has been adopted in order to support wrapping legacy system and fast adaptation to business change, etc.
- **Decision D02: Integration of Heterogeneous Information Sources.** Mediator-wrapper solution[13] has been applied to integration of heterogeneous information sources.
- **Decision D03: Role of Mediator and Wrapper.** The mediator aggregates different data sources, and provides unified representation of business data. When the mediator receives a query from a user, it forwards the query to data sources. The mediator also aggregates results from data sources, and provides an aggregated result to the user. Next, the wrapper converts queries over information in the common model into requests that the source can execute, and it converts the data returned by the source into the common model.
- **Decision D04: Common Data Model.** The decision is made to use global metadata repository, which provides global schema as data standards and mapping information between local schema and global schema.
- **Decision D05: Range of Data Share.** It is very important to decide the range of data share between the NTIS and each R&D center's system. Replicable data from each R&D center are previously duplicated to common DB of the NTIS due to the performance, and non-replicable data are transferred to the NTIS through connection of systems. The consistency problem between replicated data and original data is important, but it is out of consideration in here.

- **Decision D06: Network Configuration.** The decision is made to setup networks between the NTIS and R&D centers' systems. There are two alternatives for network configuration. One is based on existing public network, and the other is based on the private network which is set up using some hardware or software. Currently the public network solution has been chosen.
- **Decision D07: Location of Wrappers.** The location of wrappers is largely related to maintenance issue such as updating schema. There are two choices for wrappers' location. One is to locate on the NTIS, and the other is to locate on R&D centers. Currently wrappers are located on the NTIS.
- **Decision D08: Communication between the NTIS and R&D centers' systems.** The decision is made to use XML data representation for the information and use common HTTP(S) protocol for the transportation. Also, the NTIS provides SOAP/HTTP way to manage the behavior of each wrapper.
- **Decision D09: Connection with Legacy Systems.** The construction of the NTIS faces with the challenges having to connect legacy systems. To do this, a method wrapping legacy systems as services is applied to the NTIS.
- **Decision D10: Access Points.** The decision is to set up access point in R&D centers. It is necessary not to have an impact on R&D center's local system. The access points provide interfaces for accessing data of R&D centers to the NTIS. Also, the access point includes a data query component to generate the XML data.
- **Decision D11: Information Services.** The functions for accessing R&D project data are exposed as information services. Through using information services, R&D project data can be accessed without knowing details about data access.
- **Decision D12: Application Services.** The applications are also exposed as services on ESB. They are reused and interacted under the control and management inside ESB.
- **Decision D13: Multi-channel Presentation.** The portal solution has been applied to the NTIS for supporting customization of user interfaces to different users.

B. Evaluating the Decisions

In the architecture of Figure 6, all the R&D centers are connected to the NTIS through public network according to the decision D06. In addition, the information gathered from them is accessible using HTTP(S) protocol. Namely, the NTIS uses XML data representation for the information, and uses common HTTP(S) protocol for its transportation. Therefore, each R&D center uses a data query component to generate the XML data and uses some wrappers to expose them through HTTP protocol. The data generator (i.e. data query component) will not care about transformation and verification, so it won't need to be changed when the data schema of R&D center is changed. Also, the NTIS provides some control functionalities through SOAP/HTTP way to manage the behavior of each wrapper. The data transformation is done at the NTIS site, which is under control of the NTIS. Wrappers at the NTIS site provide the uniform data format for the information services. However, current decisions (decision D06 through decision D08) provide benefits of system maintenance to the change of R&D centers, but there is performance loss due to using XML/HTTP(S).

C. Changing the Decisions

If we would like to set up the private network with all the R&D centers, it could be possible using commercial product to help data integration. The data integration product such as Websphere Information Integrator[14] will take the responsibility of data gathering, synchronization from different R&D centers directly at the database level. It will have great performance advantages over the method of using XML/HTTPS in many fields, such as security, performance, easy management, etc. However, before applying another architectural alternative to current architecture, it is necessary to perform a change impact analysis. To do this, we first generated a decision constraint graph through using constraint relationships among decisions. For example, the decision D06 affects the decision D08 about a communication way between the NTIS and R&D centers' systems. It also affects the decision D10 about the access points. Therefore, the decision D06 has relationships with both the decision D08 and the decision D10. In this way, the decision constraint graph can be generated through analyzing relationships among decisions as shown in Figure 7(a). Next, it is necessary to gradually traverse nodes of the graph according to required changes by checking consistencies by relationships among decisions. Figure 7(b) represents the sequence of tracing changes, and Figure 7(c) represents nodes requiring changes. Therefore, we can track arbitrary changes according to the dependencies between design decisions the architect is interested in. We have to alter some or all dependent decisions (D08, D10, and even D02, D03, D07) by changing the decision D06. Finally, Figure 8 represents the changed architecture of the NTIS.

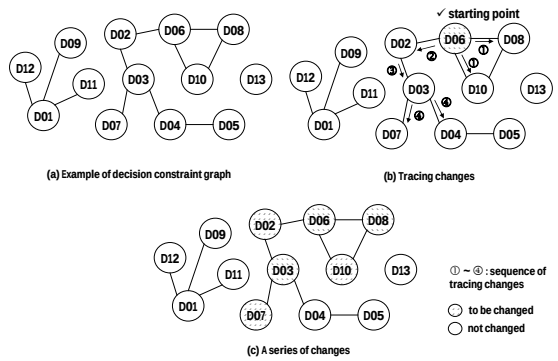

Figure 7. Change impact analysis using decision constraint graph

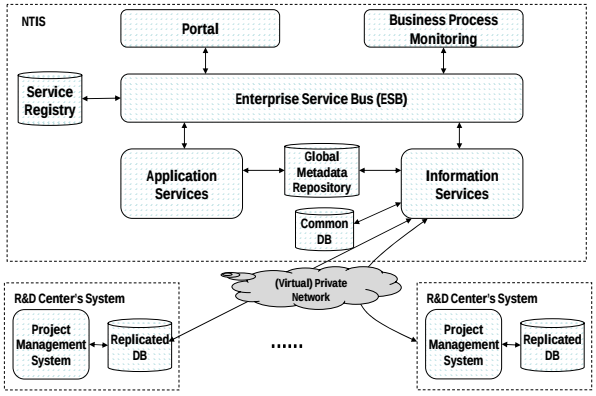

Figure 8. The changed architecture of the NTIS

VI. Comparison with Existing Methods

We compared the AQUA with the existing methods in various respects. To identify comparison elements among methods, we have studied the existing methods including previously developed comparison frameworks for evaluation methods[14,15]. As a result, we found characteristics considered to be important with respect to architectural evaluation and transformation for quality achievement. According to the characteristics, more detail comparison can be described as follows:

- **Definition of software architecture.** During acquiring good software architecture with respect to its requirements, the definition of software architecture is closely related to works for architectural evaluation and transformation. In this respect, the concept of architectural design decisions supports explicit design exploration. However, most methods lack of the concept of architectural design decisions. Compared with other methods, the AQUA defines software architecture as a set of architectural design decisions. It also presents the model of architectural design decisions.
- **Unit of focus.** Each method has the focus of different elements in its process. It characterizes methods and affects their application. Therefore, it is important to understand the unit of elements focused in the evaluation and transformation. Compared with other methods, the AQUA focuses on architectural design decisions. Through finding, evaluating, and changing the decisions, the AQUA supports quality achievement activities involving the works for architectural evaluation and transformation.

Table 1. Comparison with existing methods

Methods / Fields	SAAM [13,14,15]	ATAM[13,14,15]	ARID[13,15]	Carriere's Approach[16]	Krikhaar's Approach[17]	AQUA
Definition of software architecture	Left to users	Left to users	Left to users	Left to users	Left to users	Defined as a set of architectural design decisions
Unit of focus	Modules, Scenarios	Architectural design decisions, Quality scenarios	Architecture as Conceptual design	Architecture itself, Code	Design entities (layers, components, modules, functions)	Architectural design decision
Process support	Not explicitly addressed	Comprehensively covered	Sufficient process support	Steps explained but insufficient guidance	Steps explained but insufficient guidance	Framework to support evaluation and transformation
Method's Activities	Evaluation	Evaluation	Evaluation	Transformation	Transformation	Evaluation and transformation
Method's goals	Risk identification, Suitability analysis	Sensitivity & Trade-off analysis	Validating design's viability for insights	Transforming code for satisfying qualities	Making changes to the architecture for quality achievements	Acquiring good software architecture for quality achievement
Techniques	Scenario-based functionality and change analysis	Questioning and measuring	Combination of scenario-based and active design reviews	Reengineering	Reengineering	Decision-centric approach
Quality attributes	Mainly modifiability	Multiple quality attributes	Suitability of the design	Multiple quality attributes	Multiple quality attributes	Multiple quality attributes
Traceability	Partially supported by scenarios	Sensitivity points and trade-off points	Not supported	Not supported	RPA model (partially used)	Model of architectural design decisions , which includes a decision constraint graph

- **Process support.** Architecture-based quality achievement requires a number of activities at the architectural level. It also needs a number of inputs, and generates important artifacts. To make the efforts successful, well-defined process is needed. Compared with other methods, the AQUA provides an integrated approach to architectural evaluation and transformation centering on design decisions.
- **Method's activities.** There are a number of activities to be handled at the architectural level for quality achievement. In particular, the activities for architectural evaluation and transformation include identifying desired qualities, specializing quality attributes, evaluating architectures, analyzing change impacts, modifying architectures, and validating changed architectures. Compared with other methods, the AQUA supports all the above activities for quality achievement in its process.
- **Method's goals.** Software architecture evaluation or transformation can be performed for a number of purposes, e.g., risk assessment, trade-off analysis, quality prediction, architectural changes and so forth. The AQUA intends to acquire good software architecture through making changes to the architecture as well as evaluating the potential of the designed architecture to facilitate the achievement of the required quality attributes.
- **Techniques.** A wide range of techniques have been applied with respect to methods' goals. In order to select an appropriate method, it is important to know which techniques are included, what level of information required, and which stage is the most appropriate to apply the method. A good method should explicitly answer these questions. Compared with existing methods, the AQUA uses decision-centric techniques in its process. It requires software architecture and its requirements. It is also applied to architectural evaluation and transformation in the early stage.
- **Quality attributes.** Software architecture has relationships with a number of quality attributes. Different communities have developed the methods to evaluate or transform software systems with respect to their respective quality attributes. For example, the ATAM focuses on architectural decisions that affect (positively or negatively) one or more quality attributes, which are called either sensitivity or trade-off points. In this respect, the AQUA is also the method for any quality attribute.
- **Traceability.** It is important to clarify relationships between design decisions and requirements, relationships between design decisions and architecture, or relationships among decisions themselves. This helps the architect with obtaining a better understanding of the software architecture. It also allows tracing impacts of architectural changes for quality achievement. The AQUA provides the model of architectural design decisions, which includes a decision constraint graph representing relationships among decisions.

Table 1 summarized the result of comparison between existing methods and the AQUA. As presented in Table 1, the existing methods have some limitations in many fields. Compared with the methods described in Table 1, however, the AQUA easily supports integration of architectural evaluation and transformation for quality achievement with architectural design decisions. More concretely, the AQUA proposed the model of architectural design decisions for making architectural design decisions more explicit. Based on the proposed model, the AQUA defined decision-centric process of finding, evaluating, and changing the decisions. To be important, the AQUA involved works of architectural evaluation and transformation during the decision-centric process. Therefore, the AQUA is characterized as an integrated approach to acquiring good software architecture for quality achievement at the architectural level.

VII. Conclusion and Future Work

Good software architecture results from making good architectural design decisions and evolving them. The works for acquiring good software architecture need to be centered on architectural design decisions with respect to its software requirements. However, the effort for acquiring good software architecture currently lacks of interest and experience in architectural design decisions. Moreover, decision-centric process for quality achievement is not well supported by existing approaches. This paper proposed the model of architectural design decisions for making architectural design decisions more explicit. Based on the proposed model, this paper also proposed an integrated approach for acquiring good software architecture with respect to its requirements, which is called AQUA. Namely, the AQUA defined decision-centric process of finding, evaluating, and changing the decisions. During the decision-centric process, the AQUA involved works of architectural evaluation and transformation. Therefore, the AQUA easily supports the integration of architectural evaluation with architectural transformation. Architectural evaluation plays a significant role in revealing any potential defects or assessing the fulfillment of required quality requirements during finding and evaluating decisions. Architectural transformation plays a significant role in reducing defects in the architecture or making changes to the architecture during changing decisions. The contribution of our work has threefold. First, we presented a model of architectural design decisions for making architectural design decisions more explicit. Second, we seamlessly integrated the activities relevant to quality achievement at the architectural level, which include architectural evaluation and transformation. Third, we compared the AQUA with existing methods with respect to architectural evaluation and transformation for quality achievement. In summary, the AQUA provides software architects with a means for achieving quality attributes through acquiring good software architecture at the architectural level.

In the future, we will document architectural design decisions more explicitly. Then we will improve the proposed model of architectural design decisions. Finally, we will evolve our approach centering on the model of architectural design decisions for tool support.

REFERENCES

[1] Jansen, A. and Bosch, J., "Software Architecture as a Set of Architectural Design Decisions", Proceedings of 5th *IEEE/IFIP Working Conference on Software Architecture*, November 2005, pp.109-120.

[2] Tyree, J. and Akerman, A., "Architecture Decisions: Demystifying Architecture", *IEEE Software*, March/April 2005, pp.19-27.

[3] Bosch, J., *Design and Use of Software Architectures*, Addison Wesley, 2000.

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

[5] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., and Stafford, J., *Documenting Software Architectures, Views and Beyond*, Addison Wesley, 2002.

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

[7] IEEE Computer Society, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE Std. 1471-2000, October 2000.

[8] Kruchten, P., "The 4+1 View Model of Software Architecture", *IEEE Software*, Vol. 12, No. 6, November 1995, pp42-50.

[9] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns*, Addison Wesley, 1995.

[10] Buschmann, F., et al., *Pattern-Oriented Software Architecture – A System of Patterns*, John Wiley & Sons, 2000.

[11] Schmidt, D., Stal, M., Rohnert, H., and Buschmann, F., *Pattern-Oriented Software Architecture – Patterns for Concurrent and Networked Objects*, John Wiley & Sons, 2000.

[12] Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., and Widom, J., "The TSIMMIS Project: Integration of Heterogeneous Information Sources", *Proceedings of 10th Anniversary Meeting of the Information Processing Society of Japan*, pp.7-18, Tokyo, Japan, 1994.

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

[14] Dobrica, L. and Niemela, E., "A Survey on Software Architecture Analysis Methods", *IEEE Transactions on Software Engineering*, IEEE Computer Society, Vol. 28, No. 7, July 2002, pp.638-653.

[15] Babar, M.A., Zhu, L., and Jeffery, R., "A Framework for Classifying and Comparing Software Architecture Evaluation Methods", *Proceedings of the 2004 Australian Software Engineering Conference*, 2004, pp.309-318.

[16] Carriere, S.J., Woods, S., and Kazman, R., "Software Architectural Transformation", *Proceedings of the 6th Working Conference on Reverse Engineering*, October 1999, pp.13-23.

[17] Krikhaar, R., et al., "A Two-phase Process for Software Architecture Improvement", *Proceedings of the International Conference on Software Maintenance*, August 1999, pp.371-380.

Heeseok Choi was born at Busan, Korea in November 1972. He received the BS and MS degrees in Computer Engineering from Pusan National University in 1998 and 2000, respectively. He was a PhD student at Department of Computer Engineering of Pusan National University from 2000 to 2003.

From 2004 to 2005, he was with Electronics and Telecommunications Research Institute. Currently, he is a senior researcher of NTIS organization at the Korea Institute of Science and Technology Information in Daejeon, KOREA. His current research interests include software architecture, service oriented architecture, component-based software development, information integration, sensor network application, etc.

Mr. Choi is a member of the Korea Information Science Society.

Youhee Choi was born at Busan, Korea in February 1977. She received the BS and MS degrees in Computer Engineering from Pusan National University in 1999 and 2001, respectively.

From 2001 to now, she has been a researcher of Embedded S/W Research Division at the Electronics and Telecommunications Research Institute in Daejeon, KOREA. Her current research interests include software product lines, software architecture, embedded software, software reuse, service oriented architecture, etc.

Ms. Choi is a member of the Korea Information Science Society.

Keunhyuk Yeom was born at Incheon, Korea in February 1962. He received a bachelor's degree in Computer Science and Statistics from Seoul National University in 1985, the MS and PhD degrees in Computer and Information Science and Engineering at the University of Florida in 1992 and 1995, respectively.

From 1985 to 1990, he was with LG Electronics Research Institute. After he got the MS and PhD degrees, he worked with SamsungSDS as an advisory engineer. Currently, he is an associate professor of computer science and engineering at the Pusan National University in Busan, KOREA. His current research interests include software product lines, component-based software development, software reuse, self-adaptive software, infrastructure for RFID solutions, etc.

Prof. Yeom is a member of the ACM, the IEEE Computer Society, and the Korea Information Science Society.