Modeling and Documenting the Evolution of Architectural Design Decisions

Rafael Capilla[1] Francisco Nava[1] Juan C. Dueñas[2]
rafael.capilla@urjc.es francisco.nava@urjc.es jcduenas@dit.upm.es
[1]Department of Computer Science, Universidad Rey Juan Carlos, Madrid, Spain
[2]Department of Engineering of Telematic Systems, Universidad Politécnica de Madrid, Madrid, Spain

Abstract

All software systems are built as a result of a set of design decisions that are made during the architecting phase. At present, there is still a lack of appropriate notations, methods and tools for recording and exploiting these architectural design decisions. In addition, the need for maintaining and evolving the decisions made in the past turns critical for the success of the evolution of the system. In this research paper we extend a previous work to detail those issues related to the evolution of architectural design decisions.

1. Introduction

From the fact that software changes and evolves over time, most software engineering activities are immersed in a decision making process to meet the requirements specification. Sometimes the expectations of a system cannot be satisfied because we make incorrect decisions. In other cases, architecture erosion or non existent designs lead to highly cost maintenance processes. The lack for having recorded architectural design decisions and the rationale that motivated them impedes to replay the decisions made before. As mentioned in [15], software architects make decisions everyday, but most architectural and development processes don't document these decisions explicitly.

Otherwise, making the right decisions will influence positively on maintenance activities, and such decisions can be reused for similar design problems. The impact for recording and documenting explicitly tacit architectural knowledge (AK) clearly benefits software engineers, which may increase their own expertise by re-using already proven decisions. For instance, novice software architects might be interested to learn about the right and wrong decisions made during the construction of a particular system. As the system evolves, we will be able to know which decisions were right and which ones were wrong, and mark right decisions as reusable knowledge for the future.

Codifying such architectural knowledge may serve for learning and assessment processes, so it can be used to improve the evolution tasks performed over a particular system. To achieve this, specific information should be recorded for each design decision, because in other cases the decision will be useless for the future. In this work we will focus on the characterization of AK, and in particular of architectural design decisions, under an evolution context.

The remainder of this paper is organized as follows. Section 2 describes how we modeled architectural design decisions of a real system using the ADDSS (**A**rchitecture **D**esign **D**ecision **S**upport **S**ystem) tool, and we discuss the maintenance issues encountered during the evolution of the system. Section 3 outlines our proposal for extending the conceptual model described in a previous work to support better the evolution of design decisions. Section 4 discusses related work and section 5 provides some conclusions.

2. Design Decisions with ADDSS

In a previous work [3] we described the Architecture Design Decision Support System (ADDSS), which is a web-based tool for recording and managing architectural design decisions. ADDSS 1.0 stores the requirements of the target system, the decisions that are going to be made for a particular set of requirements and JPEG images representing the architectural products generating during the architecting phase. Also, a knowledge base of well-known patterns and styles can be stored for reuse. If a particular pattern doesn't match a potential decision, a free text description can be used to explain the decision made. ADDSS supports a basic dependency model in which two decisions can be related. We mean with this that we can model that a decision depends of another decision, but we don't describe explicitly the type of the dependency. More complex dependencies could be defined, but these are not currently supported by ADDSS 1.0. The tool generates automatically PDF

Second Workshop on SHAring and Reusing architectural
Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI'07)
0-7695-2951-8/07 $20.00 © 2007 IEEE

documents for describing decisions, architectural products, as well as trace links between requirements, decisions and architectures. We tested ADDSS in two small real projects, a multimedia system and a virtual reality application. Next subsection explains the main issues and problems encountered in one the aforementioned projects.

2.1. Design Decisions for Building a Virtual Reality System

In 2004 we developed a virtual reality system consisting of a virtual church with a virtual tour inside the church. The VR-church application belongs to a tourist information system for the cultural heritage of a small village in the north of Spain [4]. We used the ADDSS tool to re-document the architecture of the system, but explicitly recording the decisions that led to the architecture of the system. Up to 101 requirements were specified, 62 modeling requirements for the virtual scene, 28 functional requirements, 5 nonfunctional requirements and 6 hardware and platform-specific requirements. We took four iterations to develop the architecture using ADDSS and 15 design decisions were made and stored in the system. Several architectural styles and patterns (e.g.: layers, MVC) were used for making some design decisions. During the architecting activity, we considered several alternatives for some of the decisions made and we evaluated the pros and the cons for each alternative in order to select the right decision. For instance, one of the decisions was affected by an important quality requirement (i.e.: the performance for loading the database containing the virtual objects which affects the object hierarchy to be defined in the architecture), in which a simulation process had to be carried out to decide between three possible alternatives before the final decision was made. All these design decisions, their dependencies among them, and the architectural products generated during the design phase were stored in the system database. The documentation of the architectural knowledge can be general automatically on demand if needed.

2.2. Maintenance of AK

After two years, the identification of new requirements led to a maintenance process because the architecture had some problems to scale up properly. The lack of flexibility in some parts of the architecture motivated a re-architecting process to introduce more flexibility at design and implementation levels. Therefore, some decisions had to be changed according to the new requirements in order to facilitate the evolution of the system. For instance, the original system defined a predefined path for the virtual tour. This had to be modified to support non predefined paths inside and outside the church. In other case, the activation of tourist information points depending of the position of the user changed because the user has now the ability to activate them or not. The re-design of some parts of the architecture tried to avoid some existing rigid pieces (e.g.: adding new virtual effects by reducing the adaptation effort) as well as duplicate attributes and methods in already existing classes.

Twelve new requirements were defined and during the re-architecting activity, some decisions made two years ago were now considered obsolete. The current version of ADDSS doesn't support some of the features that could be useful for development and maintenance processes, as described in section 3.1. For instance, a description of the status of the decisions (e.g.: pending approved, obsolete), explicit alternative decisions, or complex relationships (e.g.: *required* or *excluded* dependencies). In other situations like a corrective maintenance, usually generated by a malfunctioning of the system, it seems important to record the motivation as well as the pros and the cons that led to a decision. Knowing this, the architect would perform easily the re-design tasks because he / she could know the reasons that led to that decision. During our experience using ADDSS 1.0, we perceived that certain attributes that characterize the design decisions would be important to be recorded, such as we discuss in next section.

3. Characterizing Evolvable AK

When designing the architecture of a software system, architects frequently retain their own expertise in their minds and only part of this is made explicit during the architecting activity. As requirements change, the decisions made for maintaining a system evolves accordingly to the requirements. The characterization of evolvable AK requires the specification of appropriate attributes that describe such knowledge. From the experience gained during with the VR-church system using ADDSS, we noticed that not all the attributes of the decision made are needed at the same time. As there is no agreement about which information should be considered relevant to describe design decisions, in next subsection we propose a characterization of the items that could be important for describing suck knowledge at different stages of the software lifecycle.

Second Workshop on SHAring and Reusing architectural
Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI'07)
0-7695-2951-8/07 $20.00 © 2007 IEEE

3.1. Relevant Architectural Knowledge

The identification of what information should be relevant for characterizing architectural design decisions is the first step before defining the process for managing such knowledge. During the architecting phase, we believe not all the items that characterize design decisions are used at the same time. Frequently, decisions made by expert architects remain implicit in the architect's mind and such knowledge is never codified. A way to achieve a balance between what information becomes relevant for codifying architectural knowledge, is to distinguish between mandatory items and optional items.

- **Mandatory attributes**: Are attributes associated to architectural design decisions necessary to be defined during the whole life of the system. The combination of all these attributes could be seen as the "rationale" that guides our decisions.
- **Optional attributes**: Are attributes that without making them explicit a design decision can be made, but they are considered quite useful for having additional complete information about the decisions we want to store. Each particular organization may decide which items belonging to this category have to be stored.

MANDATORY ATTRIBUTES

- **Decision name and description:** Contains the name and a brief description of the decision made.
- **Constraints:** Before a decision is made, constraints specified in the requirements should be taken into account in order to make a viable decision or select an appropriate choice.
- **Dependencies:** Before making a new decision we have to know if a new decision depends of another already made. Dependencies must be set during the architecting process. Is not our aim to provide a classification of dependency types (e.g.: depends of, excludes, enable, incompatible with, etc [12].) because this can be defined for each particular project or organization.
- **Status:** We must mark the status of the decision to know if a decision is still pending, rejected or approved.
- **Rationale:** It is important to record the reason why a decision is made and the rationale that guided this decision.
- **Design patterns**: Most design decisions are based on well-known architectural patterns and styles, or other design rules that provide a particular solution for a design problem.
- **Architectural solution:** We have to relate one or more decisions to a particular architectural product, which is obtained during the development of the system.
- **Requirements:** Similarly, we have to mark the requirements that led to a particular decision. In other case we will loose the origins that motivated such decision.

OPTIONAL ATTRIBUTES

- **Alternative decisions:** Expert architects usually make decisions based on a long personal experience. In such scenario, it is frequent that existing alternatives are not explicitly recorded because they reside in the architect's mind. On the contrary, recording all the alternatives considered during the development of the system seems quite useful (e.g.: in a maintenance context when an error appears or if we have contradictory or dependant decisions that makes invalid the alternative selected).
- **Assumptions:** Architects make on assumptions on which decisions rely. These assumptions and the rationale that guided them are quite useful to understand the motivation of the decisions made at the early stages. Learning activities can benefit for having recorded the assumptions.
- **Pros / Cons:** Knowing the implications of a particular decision (the pros and the cons) is of major interest for assessment and for evaluating different alternatives.
- **Category of decisions:** Each decision may belong to a particular category, which can be used for discovering purposes. For instance, a variability management tool may visualize different product configurations depending of the category of the choices made (i.e.: the variation point is used as a decision mechanism).
- **Iteration:** Because all architectures are built under an iterative process (as a software project does), it seems useful to record the architectural iteration to which the decision belongs to.
- **Project / Software architecture information:** This attribute is more project-specific and contains information about the target architecture and the project.

0-7695-2951-8/07 $20.00 © 2007 IEEE

- **Responsible**: Assumes the responsibility of the project or for a set of decisions made.
- **Architecture view:** Represents a particular view of the target architecture. We can use this attribute to search for those decisions that affect a particular view.
- **Stakeholders**: Contains the list of stakeholders interested on a particular view and hence on a particular subset of design decisions.
- **Related principles:** Are related design principles that can be interesting at a given moment.
- **Notes**: Complementary information of interest for a particular decision.
- **Quality attributes:** Contains a quality attribute that affects to one or more decisions and it only appears for decisions influenced by non functional requirements. This attribute can be used to know specific decisions that are only related to the quality features of the architecture.

The following attributes seem to be more useful during evolution activities, but some of them they should be recorded when the architecture is designed for the first time.

- **Date / Version:** Stores the date and the version of the decisions made. This is useful track the history and versions of past decisions.
- **Obsolete decision:** The attribute can be incorporated to the status of the decision and expresses legacy decisions. Older decisions could be removed out from the knowledge base.
- **Validity:** Once the system has been developed, we expend a certain period testing it in the real site. During this period we can discover errors and we can mark decisions as right or wrong.
- **Reuse times / rating:** This attribute indicates the number of times a decision has been reused. A score for the most reusable decisions can be assigned, and this may impact on new developments.
- **Trace links:** Trace links represent the connection between requirements and decisions and between decisions and architectural products. As stated in the "decision view" [6], decisions are used to establish links between requirements and architectures and vice-versa, which can be traced forward and backward. Traces are used for controlling the evolution of the system in combination with the dependencies.

Once a decision is created, we can store the information defined as mandatory and leave the optional data for subsequent refinements. Adopting this flexible approach, each organization can decides about the relevance of the attributes they need to characterize their own design decisions.

3.2. A Meta-model for Architecting and Evolving AK

To support the characterization of design decisions as well as the processes needed to manage the decision activity, we have extend the meta-model described in prior work [3] to include the items already described. Our new meta-model defines clearly three main parts.

Project Model: Comprises the information related to the software architecture project. Each stakeholder is interested on certain functional and non-functional requirements. The future architecture of the system will be developed or modified during a set of iterations. Architectures can be described by means of one or more architectural views [13] on which the stakeholders are interested. Each view must be properly documented and design decisions must be explicitly documented as part of the "decision view".

Architecture: Represents the software architecture which is mainly described as a set of components and connectors. Architectures are built using well-known patterns and they are the result of a set of design decisions. Variability is also considered as a type of decision, in particular for product-line architectures. Other architectural views may be described using other types of elements but they have not been explicitly described in the architectural model. Therefore, we have not described in the architecture box how an architecture is described (e.g.: components, connectors, ports, etc), because this will depend on the specific architectural view chosen.

Decision Model: Is considered the cornerstone of the meta-model. Following the distinction mentioned in [10], it comprises two types of AK. The first type, architectural knowledge "as a product" is represented with the box *architectural design decision*, and express the way decisions are described. Decisions can be characterized with the attributes specified in section 3.1 (e.g.: status, pros and cons, validity), but for clarity most of them are not depicted in figure 1. In addition, decisions are connected to the "decision view" in the project model because they have to be documented explicitly. All the decisions we make explicit for a particular architecture (i.e.: alternative decisions and

those who are selected as the right decision) belong to the architectural design decision box. Decisions are driven by the requirements, so an explicit connection between them is defined. Design decisions are the result of a decision making activity enacted by the architect. Finally, dependencies and constraints strongly influence the decisions made by the architect. Trace links are defined on the basis of dependencies already established during the decision activity.

The second type of AK is architectural knowledge "as a process", which is represented by the *decision making process* box. Under this category we specify all the activities carried out by the architects and other stakeholders to: store, share, evaluate, assess, manage, document, communicate, learn, reason, discover and reuse architectural design decisions. Each organization or tool may define and implement its own processes. The decision making activity has a direct connection to the decisions because they are the result of any of the any of the aforementioned activities. Also, decisions serve as input for some of these processes. For instance, we can evaluate several alterative decisions to provide the optimal or the right one.

organization or tool support. For instance, the attributes that mostly drive the evolution activities are not depicted at all, but they can be incorporated for a specific implementation of a particular decision model. Moreover, concrete implementations of AK may be able to use a concrete representation technique for storing the architectural design decisions (e.g.: network, ontology, etc.), but independently of the meta-model already described. Only certain activities like discovering, storing and reuse may be affected by the representation technique selected.

3.3. An Integrated View for Architecting and Evolving Design Decisions

The aim of the information described in section 3.1 and supported by the meta-model of Fig. 1 is twofold. First we have described the information that should be documented for making explicit the tacit knowledge of architects.

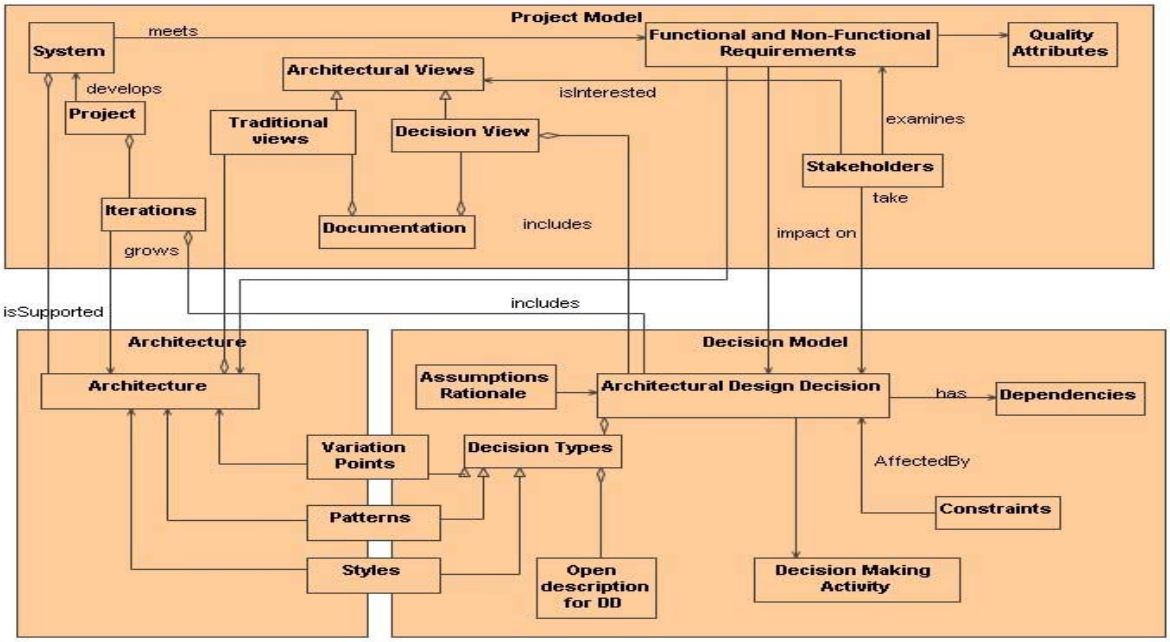

Figure 1. Meta-model for architecting, managing and evolving architectural design decisions

The meta-model of figure 1 describes three main boxes, each of them for representing the three main parts already mentioned. Some of the attributes described in section 3.1 are not represented in the figure, which contains the most relevant pieces of the model. These attributes can be associated to its corresponding box and detailed for a particular

We have used a flexible approach for documenting different types of AK attributes and for making agile the decision process. Second, we employ this characterization in conjunction with the processes that belong to any decision activity as a way to record, maintain and evolve AK. Thus, we provide an

0-7695-2951-8/07 $20.00 © 2007 IEEE

integrated view by combing the three main boxes of the proposed meta-model to describe the relationships between all the elements that use AK in the architecting process. This integrated view combines the traditional approach of architecture as a set of components and connectors with a more modern one which sees architecture as the result of a set of design decisions. Decisions, products and processes are now integrated under the perspective of different stakeholders (Fig. 2).

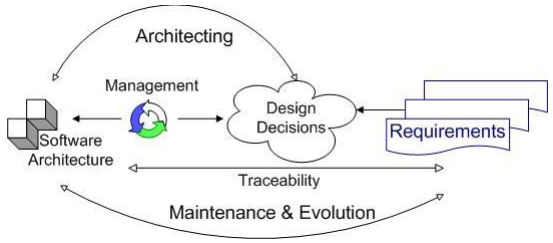

Figure 2. An integrated view of requirements, products and design decisions

4. Related Work

The traditional view of software architecture [1] is being modernizing by considering design decisions as key co-product. In [5], the importance for recording design decisions is recognized and some examples of this are given. In addition, the outcome of the discussions given in [10] suggests different types of AK: AK as a product (i.e.: design decisions, patterns, meta-models) and AK as a process.

Bosch says in [2], that we should consider software architecture as "*a composition of a set of architectural design decisions*", and he advocates for explicit representation of design decisions considered as first class entities that guide the architectural construction process. This idea has been detailed in [7] using the Archium approach, which uses a meta-model for maintaining relationships between design decisions and software architectures with some tool support. This meta-model is composed of three sub-models: an architectural model, a design decision model, and a composition model, but evolution is not explicitly addressed. In addition, our meta-model widens the use of design decisions described in [7] because it defines a variety of activities for exploiting this AK (e.g.: discovering, assessing, etc.). Also we make an explicit reference to the documentation of AK through the "decision view" of the architecture. The architecture-centric concern analysis (ACCA) method [16] employs a meta-model for capturing architectural design decisions which are linked to software requirements and architectural concerns. The authors identify the causes of architectural concerns and they assess about wrong or incorrect decisions taken. Lago and van Vliet model explicit assumptions to enrich architectural models [11]. A meta-model is derived for documenting the assumptions made in order to understand the architecture and the decisions that led to it. Tyree and Akerman [15] proposed a detailed template for capturing the rationale of design decisions in order to understand the impact of the choices made by the architects. Up to 14 relevant items are mentioned in the proposed template. The authors focus on the process by which design decisions are carried out and they provide a list of potential viable decisions. An interesting point in this approach is that they consider a network for describing dependencies in architecture decision model. By contrast, the approach suggested in [8] uses ontologies for architecting design decisions as first class entities. Kruchten et al. [9] mention the importance for building and evolving architectural knowledge and following the approach described in [8], they propose a list of attributes of architectural design decisions. A wide list of dependencies and a "category" attribute to which each decision may belong to is also discussed. This category attribute can be used for classifying, discovering and reuse AK. Finally, prior work suggests "the decision view of software architecture" [6] which crosscuts other traditional architectural views and focuses on all documentation issues relate to architectural knowledge. In addition, some data about the use and documentation of architecture design rationale has been reported in [14].

5. Conclusion

In this paper we highlight the importance for recording and using explicitly architectural design decisions. We have used the ADDSS tool for architecting a real system and we have described some of the problems encountered during the evolution of the system using ADDSS 1.0. To address these issues we have proposed a detailed template in the same way as defined in [16], but we have made an explicit distinction between mandatory and optional attributes for characterizing the design decisions. Therefore, a particular organization or architect may decide with of these attributes could be more relevant to be stored as part of their AK. Also, as systems and decisions evolve over time, we have put emphasis on attributes more closely to evolution activities. The characterization of this AK is well supported by the meta-model of figure 1, which extends and improves existing proposals by explicitly including the "decision view" and the processes that affect

0-7695-2951-8/07 $20.00 © 2007 IEEE

the decision activity. The combination of AK as a product and AK as a process in the same meta-model provides a more robust model to support and exploit this AK. Moreover, we believe that this integrated view provides a better understanding about the products and the processes that can be used for creating, exploiting and managing AK.

The iteration box of the project model defines the natural way in which systems are created and decisions are made and facilitates the creation of architectural design decisions. Architects or other relevant stakeholders should assume the liability of the decisions stored so each particular organization can benefit from this stored AK.

We are aware that ADDSS 1.0 doesn't support many important features described in the paper, but we are working on version 2.0 to introduce much more characteristics. The current status of ADDSS 2.0 includes some modules we have already developed, but they haven't been tested yet in a real system. These new modules include, among others, the following: *support for different architectural views, categorization of patterns and styles, support for different types of stakeholders, PDF documents improved, and explicit support for alternative decisions and for the status of the decision*. We are still working on a query subsystem for knowledge discovery and reuse that we expect to be finished in next months.

6. References

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

[2] Bosch, J. Software Architecture: The Next Step, Proceedings of the 1st European Workshop on Software Architecture (EWSA 2004), Springer-Verlag, LNCS 3047, pp. 194-199 (2004).

[3] Capilla, R., Nava, F., Pérez, S. and Dueñas, J.C. A Web-based Tool for Managing Architectural Design Decisions, Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge, ACM Digital Library, Software Engineering Notes 31 (5) (2006).

[4] Capilla, R. and Martinez, M. Software Architectures for Designing Virtual Reality Systems, 1st European Workshop on Software Architectures (EWSA), Springer-Verlag LNCS 3047, pp. 135-147, (2004).

[5] Clements, P., Bachman, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. and Stafford, J. Documenting Software Architectures. Views and Beyond, Addison-Wesley (2003).

[6] Dueñas, J.C. and Capilla, R. The Decision View of Software Architecture, Proceedings of the 2nd European Workshop on Software Architecture (EWSA 2005), Springer-Verlag, LNCS 3047, pp. 222-230 (2005).

[7] Jansen, A. and Bosch, J. Software Architecture as a Set of Architectural Design Decisions, 5th IEEE/IFIP Working Conference on Software Architecture, pp. 109-118, (2005).

[8] Kruchten, P., Lago, P., van Vliet, H. and Wolf, T. Building up and Exploiting Architectural Knowledge, 5th IEEE/IFIP Working Conference on Software Architecture, (2005).

[9] Kruchten, P., Lago, P., and van Vliet, H. Building up and Reasoning about Architectural Knowledge, 2nd International Conference on Quality of Software Architectures (QoSA), pp. 43-58, (2006).

[10] Lago, P. and Avgeriou, P. First Workshop on Sharing and Reusing Architectural Knowledge, To appear in ACM Software Engineering Notes, ACM SIGSOFT Software Engineering Notes, 3(5), 32-36 (2006).

[11] Lago, P. and van Vliet, H. Explicit Assumptions Enrich Architectural Models, International Conference on Software Engineering (ICSE'05), 206-214, (2005).

[12] Lee, K., and Kang, K. Feature Dependency Analysis for Product Line Component Design, International Conference on Software Reuse, LNCS 3107 Springer-Verlag, pp. 69-85, (2004).

[13] Rozanski, N. and Woods. E. Software Systems Architecture: Working with Stakeholders Using viewpoints and Perspectives, Addison-Wesley (2005).

[14] Tang, A., Babar, M.A., Gorton, I. and Han, J.A. A Survey of the Use and Documentation of Architecture Design Rationale, 5th IEEE/IFIP Working Conference on Software Architecture, (2005).

[15] Tyree, J. and Akerman, A. Architecture Decisions: Demystifying Architecture. IEEE Software, vol. 22, no 2, pp. 19-27, (2005).

[16] Wang, A., Sherdil, K. and Madhavji, N.H. ACCA: An Architecture-centric Concern Analysis Method, 5th IEEE/IFIP Working Conference on Software Architecture, (2005).

0-7695-2951-8/07 $20.00 © 2007 IEEE