A Web-based Tool for Managing Architectural Design Decisions

Rafael Capilla, Francisco Nava
Universidad Rey Juan Carlos
c/ Tulipán s/n
28933 Madrid, Spain
+34 914 888 119 / +34 916 647 489
rafael.capilla@urjc.es
francisco.nava@urjc.es

Sandra Pérez
Universidad Rey Juan Carlos
c/ Tulipán s/n
28933 Madrid, Spain
+34 610 212 242
sperezd@gmail.com

Juan C. Dueñas
Universidad Politécnica de Madrid
Ciudad Universitaria s/n
28040 Madrid, Spain
+34 913 366 831
jcduenas@dit.upm.es

ABSTRACT

Software architectures represent the design for describing the main parts of a software system. In software projects, different stakeholders with different roles may need to share the documentation generated during the project. Also, during the architecture construction phase we need to communicate the architecture to the stakeholders involved in the process, but the lack of tools for documenting, managing and sharing this architectural knowledge constitutes a big barrier. In other cases it can be useful to recreate the design decisions taken because such decisions are frequently lost during the development process. To cover these issues, we outline in this paper a web-based tool able to record and manage architecture design decisions.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]

General Terms

Management, Documentation, Design.

Keywords

Software architecture, architecture design decisions, software patterns, requirements, traceability.

1. INTRODUCTION

From the traditional point of view, software architectures have been considered as a set of interrelated components and connectors [1] [2] [20]. Under this premise, the functionality of a software system is mainly described by means of a set of software components which are inter-related by appropriated connectors.

The functionality depicted in any architecture should reflect the requirements specified in the analysis phase. Moreover, non-functional requirements drive the quality of the architecture and restrict the shape of the final design. In addition to this, the construction of software architectures is based on the use of well known design patterns and architectural styles. The rationale that guides the selection of patterns and style constitutes the main decisions taken during any architectural construction process.

The early work of Perry and Wolf [18] mentions the rationale and principles that guide the design and evolution of software architectures. This rationale has to be taken into account when we adopt an architecture-centric approach, but the main goal for representing the design decisions is to bridge the gap between software requirements and architectural products.

During the last main conferences and workshops on software architectures (EWSA2005 and WICSA2005), the experts and attendees established the need for recording and managing the design decisions in order to include them as part as the architecture documentation. The fact that design decisions are sadly forgotten during the design process and the need for recreate them when designs do not exist motivates this need. The ability to record the decisions taken before allows us to recreate the overall construction process or at least to perform architecture recovery processes in a maintenance context. Therefore, the importance for documenting architecture design decisions is currently being recognized by experts in the field.

The structure of the paper is the following. Section 2 describes the decision view compared to other traditional architectural views. Section 3 outlines some existing approaches for implementing design decisions. Section 4 describes our approach for implementing a tool for recording and managing design decisions. Section 5 describes the evaluation performed with two case studies. Section 6 provides the conclusions and future work.

2. ARCHITECTURAL VIEWS

The preliminary work of Kruchten [12] distinguishes “4+1” views for describing an architecture from different viewpoints. Depending of the interest of each stakeholder involved in the design process, each view represents different perspectives to describe the architecture of a system. Kruchten [12] proposed the following views:

- **Logical view**: Represents an object-oriented decomposition of the design supporting the functional requirements of the future system.
- **Process view**: Represents the concurrency and synchronization aspects of the design and some non-functional requirements. Distribution aspects and processes (i.e.: executable units) of the systems as well as the tasks are represented in the process view.
- **Physical view**: Represents the mapping of the software onto hardware pieces. Non-functional requirements are represented in this view and the software subsystems are represented through processing nodes.
- **Development view**: Represents the static organization of the software in its environment. The development architecture view organizes software subsystems into packages in a hierarchy or layers. The responsibility of each layer is defined in the development view.
- **Use case view**: Represents the scenarios that reflect the process associated to a set of system's requirements. This view is redundant to the previous ones (+1 in the model) but it serves to discover architectural elements and for validation purposes.

The correspondence between views can be performed to connect elements from one view to another.

In [4], the authors propose a different classification for architectural views for documenting software architectures. These views are called *viewtypes* and they define the types of elements and relationships used to describe the architecture from a particular point of view or perspective. Rather than defining new architectural views, the authors [4] modernize the description of architectural views in order to produce a more complete and accurate documentation. In addition, they state the need to record the rationale of the design decisions as part of the information needed when documenting software architectures. One weak point is that they do not mention how to record these design decisions and how they can be (re)used afterwards if needed.

A more recent approach can be found in [19], where the authors perform a nice description of views and viewpoints which constitute an approach to structuring the architectural description based on the separation of concerns. The authors introduce the concept of *perspectives* as a complementary concept to viewpoints, which contain proven architectural knowledge to structure the architecture definition process but focusing on cross-structural quality properties. An updated catalog of six core viewpoints is described in detail. Finally, other approaches for classifying architectural views can be found in [7] [25] and also in [11], where a viewtype of the architecture is related to aspects.

2.1 The "Decision View"

Complementary to the approaches described before, a new architectural view is proposed in [6]. This "new" view called the decision view of software architecture stresses the importance of design decisions in any architectural process. The authors mention several reasons for documenting explicitly design decisions as a new architectural view because the nature and type of information included in this view is quite different from the information documented in other architectural views. Some of these reasons refer to: design recovery process, loss of designs, documenting forward and backward traces between requirements and architectural products or even consider design decisions as a first class of architectural knowledge that should be documented explicitly. Thus, design decisions should be recorded and documented for subsequent maintenance activities or for recreating an overall design process. Connecting decisions to requirements is also an important topic to take into account because we can be able to fill the gap between requirements and architectural products using the decisions previously recorded. The proposed view constitutes an extension of Kruchten's approach (the "4+1"**+1** model) and acts as an intermediate step between requirements and the classical views. This decision view crosscuts other architectural views because design decisions affect to the information included in the rest of the views.

This "view is not described in the IEEE Std. 1471-2000 standard [8], and unfortunately tool support for this view is commonly missing. In this scenario, recording and managing decisions needs an extra investment effort that makes realistic the implementation of the decision view. In next section we describe existing approaches for implementing architecture design decisions.

3. ARCHITECURE DESIGN DECISIONS

Based on the fact that design decisions are sadly forgotten in the development process and the design rationale is usually not documented in the design process [21], the aim of the decision view proposed in [6] is to support the variety of design decisions taken during any architectural construction process. In addition, other approaches from several authors have been recently proposed advocating for this "new" topic.

Kruchten [13] mentions the use of ontologies for architecting design decisions as first class entities. The aim of this ontology is to organize the architectural knowledge and their relationships and attributes in complex-intensive systems. Similarly to Kruchten's approach, the ongoing efforts in the GRIFFIN project [23] try to develop notations, tools and associated methods for extracting, representing and using architectural knowledge (AK) that is not yet documented in the system. The project emphasizes on sharing architectural knowledge under a distributed context. Visualizing architectural knowledge is one of the important issues such as mentioned in [14].

Bosch says in [3], that the traditional definition of software architecture which considers this as a set of components and connectors can be replaced by "*a composition of a set of architectural design decisions*". Bosch advocates for explicit representation of design decisions which are considered as first class entities that guide the architectural construction process. The rationale that motivates the decisions, the constraints and the rules, are elements that should be recorded in this process. Additionally, Jansen and Bosch describe in [9] a model for supporting architectural design decisions based on a meta-model. The authors are currently developing an approach called Archium for maintaining relationships between design decisions and software architectures. Archium is a component language which extends Java for supporting components, connectors and design decisions. The prototype consists of a compiler and a supportive run-time environment.

Tyree and Akerman [22] motivate the need to take into account architecture design decisions in order to understand the impact of the choices made by the architects. The authors focus on the process by which design decisions are carried out and they provide a list of potential viable decisions. One interesting point in this approach is an architecture decision model seen as a network for describing the dependencies between decisions.

The architecture-centric concern analysis (ACCA) method described in [24] captures architectural design decisions but linking them to software requirements and architectural concerns. A traceability map captures the decisions from a set of software requirements. The authors identify the causes of architectural concerns and they assess about wrong or incorrect decisions taken. This approach is interesting in the way that requirements play an important role to motivate the decisions taken. The automation of the proposed methods is one of the lacks in their approach.

The decision view proposed in [6] mentions the process for managing design decisions. The authors describe an initial attempt for recording and visualizing design decisions. They also mention that decisions should be connected to software requirements in order to cover the overall design process. In addition, traceability issues are mentioned as an important outcome when linking requirements to architectural products. Forward and backward traces can be established once we have stored the decisions taken for a particular architecture. Moreover, the authors describe the information that can be included when recording design decisions, such as the decision number, the iteration performed, the name of the decision taken, the rationale that motivated the decision or the pattern selected. In this work we use this approach [6] as the basis for implementing a web-based tool for recording and managing design decisions such as we explain in next section.

4. A WEB TOOL FOR IMPLEMENTING DESIGN DECISIONS

In this section we describe the main contribution of this work as a continuation of the ideas presented in [6]. Our proposal describes a web-based tool for recording architectural design decisions so we can document them explicitly. These decisions can be managed and used for subsequent maintenance processes. Our approach starts building a decision network and the user (e.g.: an architect) can browse the decisions taken at the same time they are recorded.

This section is structured in three subsections. First we describe the requirements we believe a tool like this should have for the goal we pursuit. Second, we describe a meta-model as the theoretical base that supports the proposed tool. The last subsection outlines the main features of this prototype tool and how it works.

From our knowledge, this tool constitutes one of the first prototypes of the type of tools able to manage architectural design decisions. An easy web-based environment facilitates the sharing and reuse processes of architectural knowledge in distributed groups. Also, using a web interface facilitates a remote evaluation in order to improve it with the feedbacks received from people interested in the tool.

4.1 Requirements for Implementing a Tool for Managing Design Decisions

The first thing we did before starting the implementation was to define a set of characteristics and requirements our tool must support in order to make feasible the decision view described in section 2.1. In a similar way as mentioned in [5], we thought in the following needs.

- **Multi-perspective support:** This feature provides support for different stakeholders involved in an architectural project (e.g.: project managers, architects). Different kind of users may visualize and maintain the decisions stored by the tool. Support for different architectural views can be included under this requirement.
- **Visual representation**: Design decisions should be easily visualized, understood and replayed. Therefore, a simple user interface using web technologies seems to be appropriate for representing the decisions and architectures as well as the multi-perspective support. Navigation facilities should be included.
- **Complexity control**: Since in large systems the set of decisions is also large, some kind of mechanism (hierarchy, navigation, abstraction) is required in order to keep it under control. The "scalability" requirement is closely related to this one.
- **Groupware support**: This is now as an acknowledged fact that several stakeholders must interact in order to check and solve their conflicts. This feature has sense in distributed teams and offshore organizations.
- **Gradual formalization:** Because the decision making process is a learning process and thus the decisions evolve over time, a chronological visualization of the decisions taken may help for such learning process.

In addition to the features mentioned above, we have defined other useful characteristics that are closely related to the functionality of the tool. These characteristics are the following.

- **Multiple projects and architectures:** The tool must provide support for multiple projects. Each project can include one or more software architectures.
- **Different categories of users and permissions:** Related to the multi-perspective support feature, several types of users grouped under different categories can be defined. Each user will have different permissions for visualizing and managing the information stored
- **Architecture iterations:** Software architectures are obtained as a result of an iterative process. The tool should be able to define a set of ordered iterations. Iterations store one or several architectures which represent an intermediate or final product as part of the overall architectural construction process.
- **Design decisions support:** The user should be able to select one or several design decisions from those ones previously stored in the database. Design decisions previously stored can be based on well-known design

pattern or architectural styles. Users can select decisions or they can add new ones. Decisions can be described using free text.

- **Patterns & Styles:** The user can add or remove well-known design patterns and architectural styles to the database. A graphical and textual description of the pattern/style must be included. Some architects and practitioners perform a distinction between architecture patterns and styles, but this does not have influence on the rationale behind a particular decision.

- **Dependencies between decisions. Decision tree or network:** The tool should be able to relate design decisions for establishing dependencies and constraints between decisions. A decision tree or network can be used to organize the decisions taken. Decisions are numbered and linked properly so they can be shown to the user in a chronological order. Different classes of dependencies could be defined.

- **Alternative decisions:** It could be useful to provide a list of alternative decisions for the user. Wrong and right decisions can be categorized based on past experiences.

- **Architecture visualization:** Architectures can be added and visualized by the user. Once a user uploads the figure corresponding to the architecture, the system generates and stores a thumbnail image that will be visualized by the tool for navigation purposes.

- **Support for functional and non-functional requirements:** The tool provides support for functional and non-functional requirements. For each design decision one or several requirements can be selected. This allows us to connect requirements to architectural products for tracing purposes. The first version of the tool described here doesn't provide support for quality attributes and values, which are usually associated to non-functional requirements.

- **Architecture documentation:** Once the architecture is built, the tool must be able to generate a standard documentation with all the information generated during the construction of the architecture. This documentation should include images of the architectures, decisions taken, project description, and so on. Documents should be generated automatically by the tool.

Respect to non functional requirements [16], we can think of some them which can be suitable for this tool. For instance, usability, maintainability and scalability are interesting quality attributes that can be implemented. Usability concerns to an easy user interface that can be managed by several stakeholders and providing facilities for visualizing and managing the design decisions. Maintainability refers to the maintenance effort needed to maintain the tool capabilities and the existing functionality offered by this first prototype. As mentioned before, scalability is not only related to the potential growth of the decisions already stored, but also with new features that can be added to the system.

4.2 Meta-model Support for Architectural Decisions

To support the requirements described in the previous section and to provide a theoretical base of our ideas, we propose the following meta-model. This meta-model for architectural decisions represents the relations between the decisions, the architecture, stakeholders involved in different architectural views and the iterations performed in the design process. Figure 1 shows the meta-model we used for the formalization of architectural decisions. As regards the decision making process, the main concepts are: *stakeholders, decisions, requirements and architectures.* The stakeholder, taking into account the functional and non-functional requirements he/she is interested on, and after an analysis of the architectural view pertaining to his domain of knowledge, makes a decision (from the decision model) which affects the architecture. Therefore, the architecture of the system grows (or is modified) during the iterations carried out.

The decision model acts as a knowledge repository containing information about the decisions taken, the rationale for their application and the relationships between them. In figure 1 we have included some specific types of decisions related to the software architecture, such as the use of architectural styles, design patterns and architecture variation points (e.g.: if a system family engineering approach is used). Other types of decisions can be added in order to extend the proposed meta-model. In addition, constraints and dependencies between decisions are considered in the proposed meta-model. Queries for reasoning about the decisions taken or for discovering architectural knowledge are interesting features to be considered. Finally, the representation of the set of decisions in the form as a tree or a network is part of this decision model.

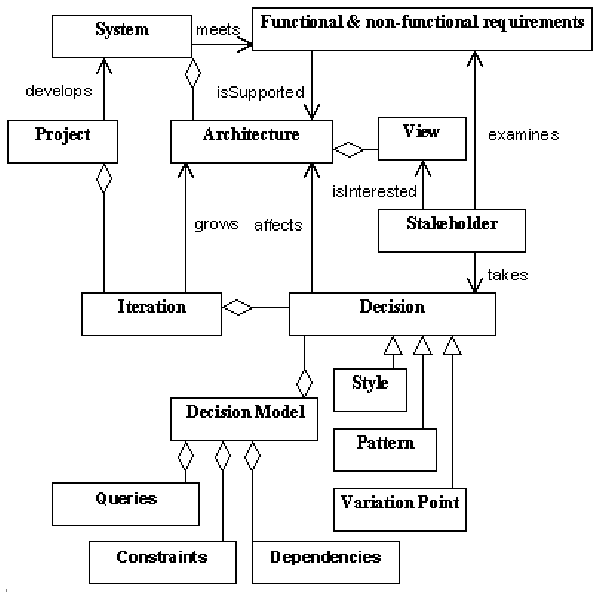


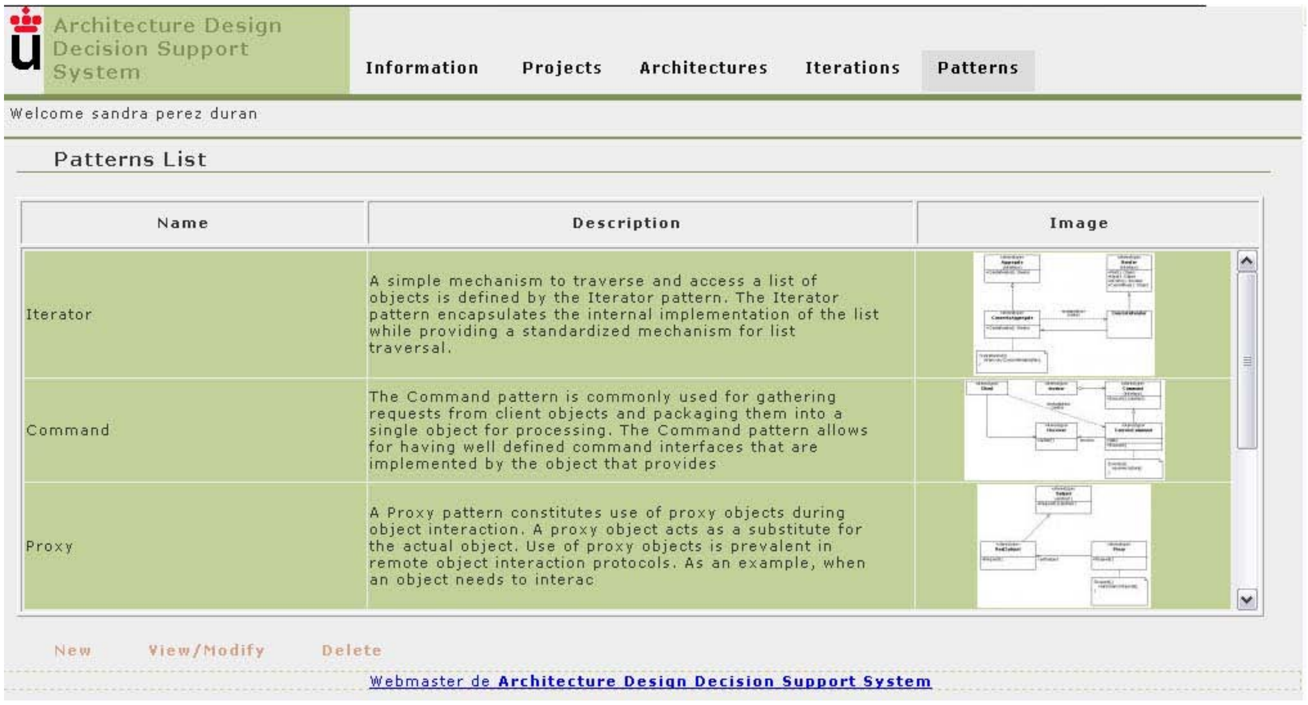
Figure 1. Meta-model for architecture design decisions.

Please notice that the meta-model of figure 1 is very focused on the process by which design decisions are formalized. This means that the iteration element plays an important role in the model because during the construction of the architecture, the user realizes the decisions as part of an iterative process.

4.3 A Web Tool for Managing Architecture Design Decisions

Based on the meta-model and the requirements described in the previous sections, we built an easy web-based tool for managing architecture design decisions. This first prototype of the tool is an ongoing project which is still being developing and evolving to include all the requirements specified in section 4.1. The architecture design decision support system (ADDSS) tool can be accessed at the URL[1] http://sevilla.escet.urjc.es/~sperez/ADDSS. The ADDSS tool is currently being developed using dynamic HTML with PHP version 4.3.10 and running under SuSE Linux 9.3 with the Apache 2 web server. The information about design decisions is stored in MySQL 5.0.16 tables. The system uses the Gd2 graphics library for producing thumbnail images of the architectures uploaded by the users. An easy web interface provides access to the functionality of the tool, such as we describe below. Different kinds of users with different roles (e.g.: project managers, architects, etc.) can be registered by filling a simple form and the system emails them a username and a password.

Registered users may have different permissions for accessing the information stored in the tool. For instance, a project manager can access the information of the whole projects belonging to the same company but a particular architect may only have access to the architectures under his / her responsibility. The ADDSS tool allows the storage of several projects and architectures. Each project may have one or several architectures. Because of architectures are built as a result of an iterative process, these iterations are described in the tool during the construction of the architecture. Users can upload figures from external files representing architectures. These architectures can be either an intermediate architectural product generated during the process and the final architecture. The decision making process allows users to make decisions based on known design patterns and styles already stored in the system. The tool provides a list of patterns such as figure 2 shows. Users can make one or several design decisions during the iterations and one or several patterns can be selected as valid design decisions. Also, those decisions that do not follow to a particular pattern or style can be added to the architecture as a textual description. As a result, all design decisions are recorded during the design process as a combination of graphical and text descriptions. The decisions made by the architect can be visualized afterwards in order to understand the rationale behind them. Some decisions may depend of other decisions and the user can establish dependencies between decisions. Thus, a decision tree or network is generated.



Welcome sandra perez duran

Patterns List

Name	Description	Image
Iterator	A simple mechanism to traverse and access a list of objects is defined by the Iterator pattern. The Iterator pattern encapsulates the internal implementation of the list while providing a standardized mechanism for list traversal.	
Command	The Command pattern is commonly used for gathering requests from client objects and packaging them into a single object for processing. The Command pattern allows for having well defined command interfaces that are implemented by the object that provides	
Proxy	A Proxy pattern constitutes use of proxy objects during object interaction. A proxy object acts as a substitute for the actual object. Use of proxy objects is prevalent in remote object interaction protocols. As an example, when an object needs to interac	

Figure 2. Pattern list stored in the ADDSS tool with a pattern description and a thumbnail image.

[1] Notice that this tool is a first prototype. Please, email us before using it because during upgrades we use a temporarily location at http://triana.escet.urjc.es/ADDSS

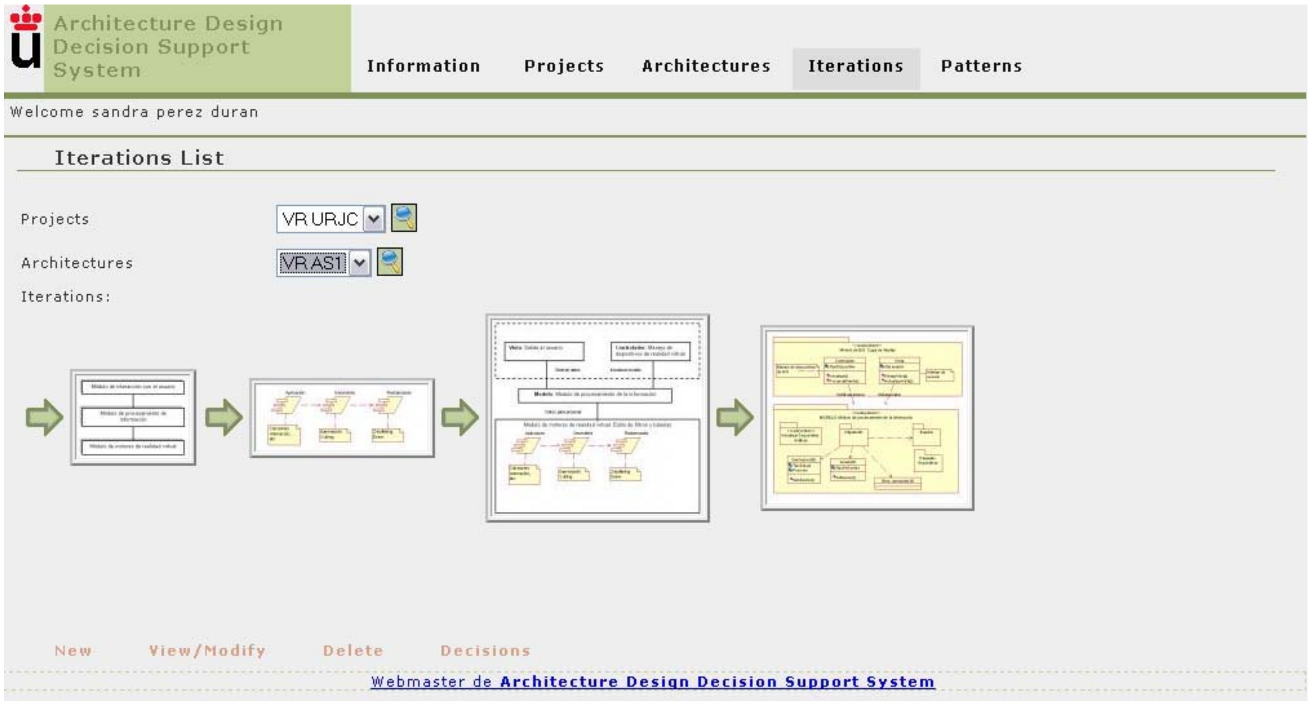
The first version of this prototype allows the connection between requirements and design decisions. In this way, users can add a list of both functional and non-functional requirements. The current implementation status of the ADDSS tool doesn't allow importing requirements from other analysis tools like Rational RequitePro, but we are planning for future versions the integration with some external analysis tools. During the design process, users can select one or more requirements and relate them to a particular design decision. Quality attributes and values are not yet supported in this first prototype. Once the user has made a set of design decisions, the system offers the possibility to visualize the decisions taken and see the chronological order of the decisions taken. For each of the iterations, a thumbnail image of the architecture is shown to the user and it can be augmented by simply clicking in the image. This thumbnail image is generated automatically when the user uploads the file containing any architectural product. In conjunction with the figures, decisions are shown to the user. Therefore, users can easily visualize the history of the architecture and the iterations performed. This approach constitutes an intuitive way by which architects can easily understand how the architecture was built and navigate through the decisions made. Figure 3 shows an example of a set of iterations which can be browsed by the user.

Initially, decisions and iterations follow a tree structure but the user has the ability to relate decisions and establish dependencies among them. In this case, the tree structure becomes a network of decisions and the tree structure will be deformed. For this first release we have only defined which decisions depend from other decisions, but we have not yet categorized these dependencies [15]. We are considering precedence and incompatibility between decisions as a first classification for dependencies.

4.4 Granularity and Hierarchy of the Design Decisions

Some interesting questions that arose during the design of the tool were the following. The first point is related to the granularity of the decisions. We mean with this the degree of the detail needed for representing a design decision. The current implementation of the tool allows us to specify decisions as design patterns, architectural styles and a free text description. This text description is an explanation of the rationale when no patterns are available. This last way for describing design decisions is useful to explain decisions that are not directly supported by particular pattern or even when the architect wants to motivate the specification of variability information, class attributes or particular relationships between architectural elements. The specification of variation points that affect to certain parts of the architecture can be explained using free text in combination with attributes and stereotypes depicted in the figure representing the design. Another situation where we can use free text happens when we need to describe cross-cutting aspects or quality attributes.

One interesting point arises when we need to define relationships between decisions. In this case we can use trees, networks or any other type of graphs for describing the structure of decisions. In our approach we used a tree or a network (depending of the cases) where the nodes represent decisions. Each node stores information about the requirements that affect a particular decision. The depth of the design decision tree (see also [10]) will depend of the number of the decisions taken and the level of detail. Finally, this information can be used to prompt the system about any particular information about the decisions made (e.g.: Which requirements affect a particular decision?).


Architecture Design Decision Support System
Information
Projects
Architectures
Iterations
Patterns
New
View/Modify
Delete
Decisions
Webmaster de Architecture Design Decision Support System

Welcome sandra perez duran

Iterations List

Projects VR URJC

Architectures VR AS1

Iterations:

Figure 3. List of iterations with thumbnail images performed for the virtual reality project.

5. PROTOTYPE EVALUATION

At present we are still evaluating the functionality implemented in the ADDSS tool. The majority of the requirements specified in section 4.1 were implemented in the tool. For evaluating the tool we stored the information belonging to two small research projects. The first project belongs to the architecture of a virtual reality (VR) system already developed at the University Rey Juan Carlos. This VR system consists in the design and implementation of a virtual church using the Multigen Paradigm tools and a virtual guide inside the church, in which context information is shown to the user as part of the 3D scene. The software architecture for this project was built in four iterations. A total of 38 requirements were identified and 28 of them were functional requirements. We employed 12 design decisions to determine the final architecture and three intermediate architectural products were obtained. In addition, we built two additional architectural products belonging to other two architectural views or viewpoints (i.e.: the deployment view and the dynamic aspects of the systems). Finally, we provided a traceability table between different architectural products and their requirements.

The second research project, developed at the Universidad Politécnica de Madrid, deals with the development of multimedia services for the domestic environment. The system under development was a personalized and interactive multimedia player, and two iterations were performed. In the first iteration the main decisions were related with the selection of the basic architectural style (publisher-subscriber), and the main architectural components. In the second iteration detailed decisions were taken about the set of codecs to be supported.

In both projects we employed different styles motivated by different design decisions. A variety of architectural products were generated during the iterations performed. Different degrees of detail of the decisions made were recorded, so they can be used for a further manipulation or maintenance process. In the first project three different views were provided while in the second one only the static view was recorded. In both projects a set of decisions related to the definition of variation points have been defined and recorded as free text design decisions.

The evaluation of the tool in both projects demonstrated us the utility for recording, visualizing and managing design decisions. To this point, the web interface was enough to browse easily the decisions taken and the iterations enacted during the architecture construction process. Therefore, the architect has a complete view of the overall design process, because the iterations shown by the tool describe chronologically the most important events happened during project. This approach facilitates the understanding of the architecture construction process for non expert users which are not directly related with the technical issues.

In addition, we generated useful documentation which includes the decisions taken, the patterns applied and the architectural products generated during the process. The tool provides three types of PDF document that are generated automatically with the information stored in the databases.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have stated the importance for documenting design decisions as a key piece in the documentation of any software architecture. Recording and documenting the architecture design decisions is quite important for both development and maintenance processes. To achieve the goals specified in the requirements, we have proposed a meta-model and a web-based tool able for recording, maintaining and managing the decisions taken during the architecture construction process. Our approach connects requirements to architectures via design decisions, so we can establish traces between them. As a result of the evaluation performed, we believe that using this kind of tools and in particular web-based tools, constitutes a new step for sharing knowledge. Offshore organizations that employ distributed teams across the world can benefit using this approach. Also, it opens new research issues for distance collaboration and for knowledge management (KM) research. In addition, the proposed tool can be used in learning processes for teaching about expertise knowledge and architecture design decisions and for understanding the implications made from right and wrong decisions. We are aware about the existence of other KM prototypes but their application in this field of knowledge is not mature enough [17]. Our approach comes from the software engineering field and is a combination between architecture construction process and architectural knowledge.

One conclusion we can extract refers to the utility of the visualization and navigation processes perceived by the stakeholders during the construction of the architecture. Users can easily browse the decisions in an ordered way and understand the rationale behind them as well as know which dependencies or constraints affect a particular decision. The facility to include basic dependencies is closely related to incompatible or dependant requirements. Another conclusion affects to the process itself. Users assisted by the tool perceived that building and visualizing iteratively the architectures, eases the construction process and the understandability becomes higher.

A third conclusion comes from the fact that using tools like the proposed one, facilitates the task of software engineers because recording the decisions made during the construction allow the architects to replay the decisions taken. This allows to recovering and reconstructing the architectures designed in the past. Also, the documentation generated in the process complements the traditional documentation obtained from other classical architectural views. Thus, knowledge can be shared and reused for new developments.

For future work some important improvements have to be implemented. The second release of the tool will include quality attributes and a detailed categorization of dependencies between decisions. Also, we will try to reinforce the support for different architectural views and how queries can be defined to discover and extract valuable knowledge from the decision network. Subsequent releases should allow the connection to other existing analysis and design tools in order to import/export requirements and architectures. Also, a guidance process for the decision making process based on proven alternatives would be of interest.

7. REFERENCES

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

[2] Bosch, J. Design and use of Software Architecture. Adopting and Evolving a Product Line Approach. Addison-Wesley, (2000).

[3] 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).

[4] 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).

[5] Dueñas, J. C. and Hauswirth, M. Hyper-linked Software Architectures for Concurrent Engineering. In Proceedings of Concurrent Engineering Europe 97, Erlangen-Nuremberg, Germany, pp: 3-10. Society for Computer Simulation. (1997).

[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] Gomaa, H. and Shin, E.. A Multiple View Meta-modeling Approach for Variability Management in Software Product Lines. Eighth International Conference on Software Reuse: Methods, Techniques and Tools. LNCS 3107, Springer Verlag, 2004.

[8] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Std. 1471-2000 (2000).

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

[10] Jazayeri, M., Ran, A. and van der Linden, F. Software Architecture for Product Families. Principles and Practice, Addison-Wesley (2000).

[11] Katara, M. and Katz, S. Architectural Views of Aspects. Proceedings of AOSD 2003, Boston, USA, ACM, pp.1-10 (2003).

[12] Kruchten P. Architectural Blueprints. The "4+1" View Model of Software Architecture, IEEE Software 12 (6), pp.42-50 (1995).

[13] Kruchten, P. An Ontology of Architectural Design Decisions, in Proceedings of 2nd Groningen Workshop on Software Variability Management, Groningen, NL (2004).

[14] 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).

[15] 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).

[16] Liao, L. From Requirements to Architecture: The State of the Art in Software Architecture Design, http://www.cs.washington.edu/homes/liaolin/Courses/architecture02.pdf

[17] Lindvall, M., Rus, I., Jammalamadaka, R. and Thakker, R. Software Tools for Knowledge Management. Fraunhofer Center for Experimental Software Engineering, Maryland, USA, http://www.dacs.dtic.mil/techs/kmse/swtools4km.pdf (2001).

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

[19] Rozanski, N. and Woods. E. Software Systems Architecture, Addison-Wesley (2005).

[20] Shaw, M. and Garlan, D. Software Architecture, Prentice Hall (1996).

[21] 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).

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

[23] The GRIFFIN project, A GRId For information about architectural knowledge, http://griffin.cs.vu.nl/

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

[25] Woods, E. Experiences Using Viewpoints for Information Systems Architecture: An Industrial Experience Report. F. Oquendo (Ed) Proceedings of the First European Workshop on Software Architecture, LNCS 3047, Springer Verlag (2004).