The Decision View of Software Architecture

Juan C. Dueñas[1,*] and Rafael Capilla[2]

[1] Department of Engineering of Telematic Systems, ETSI Telecomunicación, Universidad Politécnica de Madrid, Ciudad Universitaria s/n, 28040 Madrid, Spain
jcduenas@dit.upm.es
[2] Department of Informatics and Telematics, Universidad Rey Juan Carlos, c/ Tulipan s/n, 28933, Madrid, Spain
rafael.capilla@urjc.es

Abstract. Documenting software architectures is a key aspect to achieve success when communicating the architecture to different stakeholders. Several architectural views have been used with different purposes during the design process. The traditional view on software architecture defines this in terms of components and connectors. Also, the "4+1" view model proposes several views from the same design to satisfy the interests of the different stakeholders involved in the modelling process. In this position paper we try to go a step beyond previous proposals, to detail the idea of considering the architecture as a composition of architectural design decisions. We will propose a set of elements, information and graphical notation to record the design decisions during the modelling process.

1 Introduction

For years, the field of software architecture has been growing in width and depth; as key cornerstones of this evolution we could cite the discovery of architectural patterns, the agreed definition of software architecture in itself, the increasingly adopted lexical support for them (UML, for example), the generation of educated architects, the application of software architecture principles to the development of sets of systems (product lines and families), and so on. Very recently, the scope of work in the field has been widening even more by identifying quality attributes and their impact on the architecture of the systems, applying the architectures to distributed systems, and pieces in architecture that support the medium-term evolution of systems.

However, very recently the software architecture community has been facing its own limitations. The practical implementation of systems following the architectural approach proposed by this community is getting more and more complex, up to the extent of rendering the application of architectural approaches useless. Just an example of this fact is the perceived complexity (and instability) in the usage of platforms for enterprise computing; technologies such as J2EE have been available for several

* The work performed by Juan C. Dueñas has been partially undertaken by the FAMILIES project (Eureka 2023, ITEA ip00004), partially supported by the Spanish company Telvent and the Spanish Ministry of Science and Technology, under reference TIC2002-10373-E.

R. Morrison and F. Oquendo (Eds.): EWSA 2005, LNCS 3527, pp. 222 – 230, 2005.

years, but obtaining their promised benefits in practice seems still far of the average architect. Even more problems appear in the maintenance phase, due to the lack of explicit support for architectural decisions, as shown in [9].

In this position paper, we recall part of the original definitions of the software architecture [14][15], just to discover how poor has been supported one part of the architecting process. We also claim that the lack of coverage of this part of the architecture has lead to unmanageable complex architectures (such as those mentioned before); we propose to add some lexical support for this kind of key architectural information missed. At the far end of this vision, is the understanding of the architectural process as a decision making –and therefore a social and communication- process. Let us face it: building software architectures is taking design decisions but, once the architecture is there, these decisions evaporate.

2 Software Architecture Description

The software architecture of a system can be defined, using a well-known classical definition [15] as the structure of components, their relationships, and the principles and guidelines governing their design and evolution over time.

As for the representation of a system architecture composed by components and connectors, several graphical notations have been used, including UML. Also, different architecture description languages (ADL) (e.g.: ACME, C2, Wright, etc.) have been proposed and used to formalize the graphical notations describing the architecture. The need to describe the architectural products from different points of view [10] depending of the context and interests of the variety of stakeholders involved in the process has lead to define several views for each context and stakeholder. In this way, Kruchten's proposal [13] defines "4+1" views representing different viewpoints. These viewpoints shown in figure 1 are the following:

- **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 but it serves to discover architectural elements and for validation purposes.

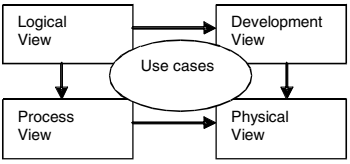

Fig. 1. The "4+1" view model (Kruchten)

The correspondence between the views of figure 1 can be performed to connect elements from one view to another. There are other classifications of views to be taken into account, and some of them have received widespread attention by the community of practitioners in the area (see [8] and [17]). In addition to this, other authors [12] propose a viewtype of the architecture associated to aspects. These authors introduce a conceptual model called *aspect architecture* which is considered as a software architecture viewtype. They propose a new UML diagram type called "concerns diagram" for modelling architectural views of aspects. Finally, in [5] the authors mention a new classification for architectural views called *viewtypes* for documenting purposes. A viewtype defines the types of elements and relationships used to describe the architecture from a particular point of view or perspective. More than defining new architectural views, they [5] try to modernize and make clear for the stakeholders the documentation generated during the architectural construction process. Also, they mention the need to record the rationale of the design decisions as part of the information needed when documenting software architectures but they do not mention how to record these design decisions in order to be used afterwards if needed.

The architectural construction process involves several elements and aspects for which the resultant software architecture constitutes the most visible part of the overall design process. Software projects involve several actors or stakeholders during the project lifecycle and the "view" of these stakeholders is quite different for each them. Therefore, the need to represent different views or viewpoints at the design level is a usual task [8].

There are many situations such as: the loss or non-existence of designs, reengineering legacy systems, evolution of architectural products, or even changes in the development team; in which it is mandatory to record the design decisions from which the software architecture was obtained at a first instance. Design decisions represent the cornerstone to obtain suitable software architectures because they represent the rationale that motivated the election of architectural patterns and styles, the functional blocks that represent systems and subsystems in the architecture, the relationships among them and the control of the architecture. Our position in this paper and following recent proposals [3] is to modernize the concept of software architecture making the design decisions explicit, and adding them a "new" viewpoint respect to the traditional approaches. Our proposal tries to detail the representation of this decision view in the architectural construction process. The traditional views of software architecture provide information enough to understand the pieces of the system under development, and also information useful to trace the requirements and features the system must fulfill, but, to date, there is no information about why a certain component or

connector has been chosen, nor why other similar elements have been rejected in the architecture. Is this information about the "why" what we try to represent in the decision view.

3 Requirements for the Decision View in Software Architecture

Again on the software architecture definition, we have seen that the structure of the system's components and relationships is described using the traditional architectural views. Also in the software architecture there are "principles and guidelines" that are out of the scope of the traditional architectural views. Since the information about principles and guidelines is still required to create the system, current practice is to create natural language documentation (or even worse, keep the knowledge on the architects' minds).

But even for these less traditional topics there are proposals worth taking into account, although most of them come from the domain of requirements engineering, such as the NFR (non-functional requirements) framework proposed by Chung [4], that characterizes stakeholders and their relationships in order to structure the decision making process launched by the tradeoffs between conflicting quality requirements. Also, in this track we can allocate the well-known ATAM method [2] and derivatives. These methods are practical enough to be used in industrial settings; however, these methods focus on the decision making process based on stakeholders. There are reasons important in order to understand the decisions taken as a result of these decision making processes, that are not made explicit, and therefore the reasons for decisions can not be linked to the architectural information (the other traditional views).

Using version management on the architectural views (storing the changes made on each of the architectural views) is not a complete solution; first this method was attempted some years ago and lead to so many deltas (in configuration management terminology) that the method shown to be useless; even worse, if the deltas were not annotated with the knowledge that drive the architects to the next version, it was impossible to replay the process.

Recent advances for supporting traceability between requirements and architectures can also help in solving part of the problem [16]. In fact, for those decisions that come directly from requirements affecting architectural elements in a 1:1 relationship, the approach may be useful. But for the decisions affecting to large regions of the architectural models (this is the case with architectural significant requirements [11] and some quality attributes), the traceability mechanisms introduce much more complexity and therefore render useless.

There are other methods –albeit old- that may help in describing the decisions that guide the architecting process: the design space theory, the application of Quality Function Deployment, Design Decision Trees [1][7], etc. When these formalisms and methods were first applied to software architecture, the lack of unified notation for the other architectural views was a key problem; nowadays, the problem seems to be partially solved by using UML.

Some of the requirements stated then for the support of the decision view seem applicable right now [6]:

- **Multi-perspective support** to provide support to the different stakeholders.
- **Visual representation** so the decisions can be easily understood and "replayed".
- **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.
- **Gradual formalization** because the decision making process is a learning process and thus the decisions evolve over time.

Once a lexical support for design decisions representation is found, the architecting process becomes a knowledge management process in which the product of the application of this knowledge produces the architectural models of the other views. We understand by "knowledge management process" that dealing with the explicit description of knowledge, the definition of the links from that knowledge to the organization that holds it and to any other element affected by this knowledge, and the support to the evolution of the knowledge and the links. Therefore, the process is able to explain why these elements have been chosen, which have been discarded and how this particular selection fulfills the system requirements. Some of the activities in this architectural-knowledge management process are:

- **Growth-refinement**: The design decisions are not isolated. As mentioned before, there is a gradual formalization that appears when architectural assessment activities are performed (both during the creation of the architecture and when architectural recovery and architectural conformance activities are done). The knowledge base formed by decisions is enlarged.
- **Dissemination and learning**: The knowledge base containing decisions is the key asset in order to learn the architecture process and this is precisely the point we try to illustrate at the beginning of this contribution: in order to cope with large or complex architectures, the ability to record and replay the decisions, provided by the explicit description of them is a key element.
- **Exploration-application**: the application of design decisions should get to the same architecture if the stakeholders, requirements and trade-offs are the same. Applying the same decisions on a different set of requirements would lead to a different architecture.

4 The Decision View of Software Architecture

The need to represent design decisions as a key aspect in the architectural construction process has lead us to propose a new view called the *decision view*. This new view has to be defined and represented in the architecture documentation so any of the stakeholders can use it later if needed.

Several reasons for record design decisions are: changes in the development team, design recovery needs, loss of designs, forward and backward traces between requirements and design products, etc. From our point of view, the explicit representation of design decisions becomes a key factor for building and communicating the software architecture.

Design decisions should connect requirements and architectural products in order to record and discover the rationale of the decisions taken during the design construction process. The information we believe a design decision should include for representing this using a UML notation or similar is the following:

- **Iteration Number**: Due that the software architecture is the outcome of an iterative process in which several design decisions are taken, we need to record the iteration of a particular decision.
- **Following Iteration**: It points to the following iteration in the design process, where iteration means the next step in the application of the design decisions, that renders an architectural model (maybe an intermediate model).
- **Decision Rule**: Represents the name of the decision rule taken by the architect. The motivation of the decision should be explicitly described here.
- **Decision Rule Number**: It numbers a specific decision rule
- **Following Decision Rule Number**: It points to the following decision rule and is used for tracing purposes or for tracking the decisions made.
- **Pattern / Style Applied**: Represents the pattern or style applied for a particular design decision. They are used to impose restrictions to a particular architectural element during the design construction process.
- **Associated Use Cases**: Represents the numbers or names of one or more use cases associated to a particular design decision. This is used to connect the architectural product to a set of requirements.

Figure 2 provides a graphical representation of a decision element which can be modelled employing a new UML element. This new element will serve to record and represent the design decisions with the information given above. Let us remark that, being a prospective work, the structure and lexical support for decisions is not definitive; in particular, the figure shows a sequential structure of decisions, but more complex topologies for interconnection may appear (binary trees [11] may be considered as a typical topology, although more complex decision networks may appear).

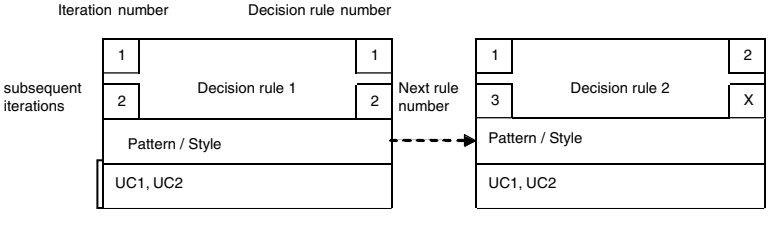

Fig. 2. Representation of the information included in the decision element

In this way, we can modify the figure proposed by Kruchten [13] to include the decision view as an intermediate element between requirements and other design views, such as figure 3 shows. The arrows in the figure indicate precedence or causality (so, for example, the decision view affects the physical view). The determination of the phase on which these views should be created is delegated to the development process. Also important to notice that this decision view is dealing with the software architecture, so it is likely that there is another decision view for requirements specification.

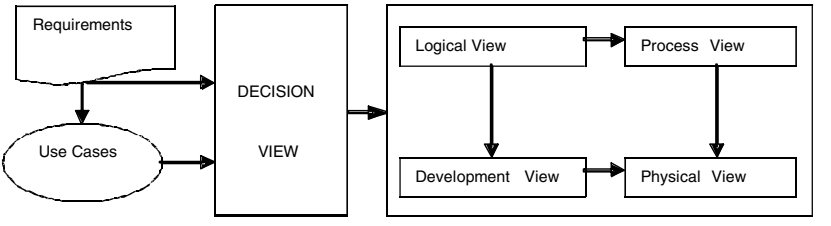

Fig. 3. The decision view model of software architecture

One key aspect when recording design decisions is how to associate these to architectural elements when we represent graphically the architectural products. For each of the iterations performed during the design process, we can assign a decision element (shown in figure 2) to each architectural element or work product. Adding backward traceability from the architectural element to the decisions that affect it may be helpful in the dissemination and learning activities mentioned in section 3. For subsequent iterations, the design decisions elements will expand to describe the rationale of the design decisions taken during the process.

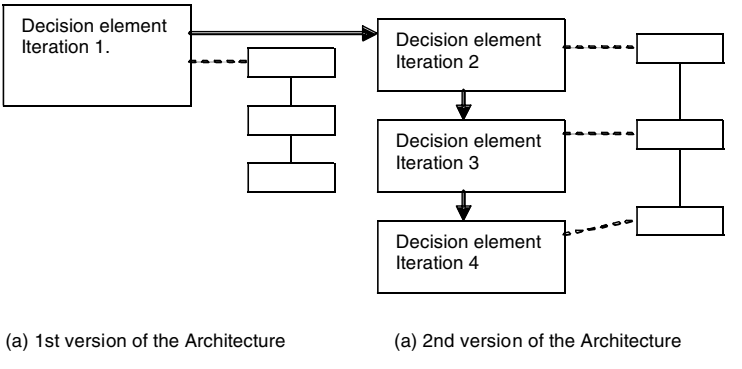

Fig. 4. Decisions elements associated to architectural elements

Figure 4 shows an example of two iterations performed during an architectural construction process. The first iteration applies a layered style for the architecture and assigns a decision element for that. The following iteration applies other architectural

styles and design decisions rules for each layer. The decisions elements shown in figure 4 are used to record and link the decisions taken.

5 A Proposal for the Implementation of the Decision View

In our proposal, so far, the key elements in the decision view are the links, or the relations between pieces of information, plus pieces of text for the rationale. The implementation of such should be the simplest if some kind of success in the industrial stage is sought. A principle just discovered in other branches of engineering may well be applied here: the quality and size of the links are more important than the qualities of the nodes. The practical application of this fact is that the decision view can be deployed as a hyperlinked documentation on top of the other views. We foresee two potential implementations for this network of knowledge: the provision of specific notations for decisions as extensions to the UML, supported by specific tools; and also the mapping of the knowledge structure to a web-based network that fosters the usage of the decision view as a communication and cooperation tool.

We are currently working towards this view; we expect that, once a knowledge base containing architectural decisions is created, and these decisions are linked between them and to the elements of the other architectural views, some activities performed in the development of the system can be supported by the navigation on that network of models. This way of understanding the software development can be called “building by browsing”.

References

1. Alonso, A., León, G., Dueñas, J.C., de la Puente, J. A.: Framework for documenting design decisions in product families development. In Proceedings of the Third International Conference on Engineering of Complex Computer Systems, Como, Italia, September (1997).
2. Bass L., Clements P. and Kazman R.: Software Architecture in Practice, 2nd edition, Addison-Wesley, (2003).
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. Chung, L., Nixon, B., Yu, E. and Mylopoulos, J.: Non-functional requirements in software engineering. Kluwer Academic Publishers, (2000)
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., 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)
7. Dueñas, J. C., León, G.: An introduction to evolution of large systems based on Software Architectures. In Systems Implementation 2000, IFIP TC2 WG2.4 Working Conference on Systems Implementation 2000, Berlin, Germay, February. Chapman and Hall, (1998) 128-139.

8. Gomaa, H., 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)
9. Graaf, L.: Maintainability through Architecture Development. F. Oquendo (Ed) Proceedings of the First European Workshop on Software Architecture, LNCS 3047, Springer Verlag, (2004)
10. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Std 1471-2000 (2000).
11. Jazayeri, M., Ran, A., van der Linden (eds): "Software Architecture for Product Families", Addison-Wesley, (2000).
12. Katara M. and Katz S.: Architectural Views of Aspects. Proceedings of AOSD 2003, Boston, USA, ACM, pp.1-10 (2003).
13. Kruchten P. Architectural Blueprints. The “4+1” View Model of Software Architecture. IEEE Software 12 (6), pp.42-50 (1995).
14. Perry, D., Wolf, A.: Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes, 17/4, October (1992)
15. Shaw M. and Garlan D.: Software Architecture, Prentice Hall (1996).
16. Stuart, D., Sull, W., Cook, T. W.: Dependency Navigation in Product Lines Using XML. Third International Workshop on Software Architectures for Product Families, F. van der Linden (ed), LNCS 1951, Springer Verlag, (2000)
17. 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)