The Decision View of Software Architecture: Building by Browsing

Juan C. Dueñas[1], 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
rcapilla@escet.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 stakeholder 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 families- of systems, 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 extend 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 are now available since several years, but obtaining their promised benefits in practice seems still far of the average architect.

Another example seems to be the increased size in elements of the Java 1.5 platform, featuring the 1.5 version of the Java language. Let us remark that the application of templates (or generics) add-ons to the core language made ISO-C++ much more complex than the average engineer is able to cope with (a demonstration of such is the relatively sparse usage of these generic elements, other than the standard template library), and this seems to the track that the Java language is following.

In this position paper, we recall part of the original definition of the software architecture, 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: making an architecture is taking decisions but, once the architecture is there, these decisions evaporate.

2. Software architecture description

Software architecture of a system can be defined, using an already classical definition [7] 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 [4] 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 [6] 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.

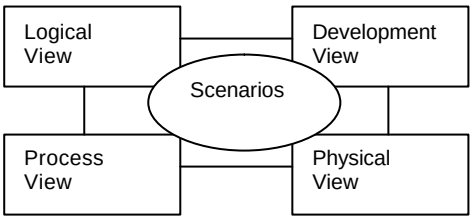

Fig. 1. The "4+1" view model (Kruchten)

The correspondence between the views of figure 1 can be performed to connect elements from one view to another. There are other classification of views to be taken into account, and some of them have received widespread attention by the community of practitioners (among these the views proposed by [13]). In addition to this, other authors [5] propose a viewtype of the architecture associated to aspects. The authors introduce a conceptual model called *aspect architecture* which is considered as a software architecture viewtype. The authors propose a new UML diagram type called a concern diagram for modelling architectural views of aspects. Finally, in [3] 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, the authors [3] 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 [13].

Although in many situations such as: lost or inexistence of designs, reengineering legacy systems, evolution of architectural products, or even changes in the

development team, lead to the need to recording the design decisions from which the software architecture was obtained at a first instance. Design decisions represent the cornerstone to obtain a suitable software architecture because they represent the rational that motivated the election of architectural patterns and styles, the functional blocks that represent systems and subsystems in the architecture, their relationships among them and the control of the architecture. Our position in this paper and following recent proposals [2] is to modernize the concept of software architecture making the design decisions explicit, and adding them as a "new" viewpoint respect to the traditional approaches. Our proposal tries to detail the representation of this decision view in the architectural construction process. So, the traditional techniques for describing a software architecture are able to give effective responses to the "how" question (let us suppose that requirements engineering is able to provide satisfactorily answers to the "what"), but, up to date, they can not describe "why".

3. Requirements for the decision view in the software architecture

Again on the software architecture definition, we have seen that the structure of the system components and relationships is described using the architectural views. The "principles and guidelines" part are out of the scope of these architectural languages, and therefore they are currently covered in a certain organization by means of out-of-band methods: written documentation using natural language in the most mature of the cases, and codified in the mind of the architects in the usual case.

But even for these less traditional topics there are proposals worth to be taken into account, although most of them come from the domain of requirements engineering, such as the NFR (non-functional requirements) framework proposed by Chung, 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 can be allocated the well-known ATAM method [1] and derivatives. These methods are practical enough to be used in the industrial setting; however, these methods focus on the decision making process and its results, but no on the documentation of the alternatives left out, the intermediate steps, nor the rationale; in this sense, the methods rely heavily on human expertise and do not address the knowledge representation problems, the "why".

What we are proposing is not the usage of versioning methods applied on the architecture models; 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 in the support of traceability between requirements and architecture can also help in solving the problem [12], but not entirely. 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 those that affect several at

once (architectural significant requirements [14]) the direct application of traceability techniques will not record all the information.

Other methods –albeit old- can help: the lines of research opened by the design space theory, the application of Quality Function Deployment, or Design Decision Trees found, in their first application to software architectures [8][10], the lack of agreements by the software architecture community regarding the lexical support that now provides UML.

Some of the requirements stated then for the support of the decision view seem surprisely applicable right now [9]:

- Multi-perspective support or, in other terms, giving support to the different stakeholders.
- Visual representation so the decisions can be understood easily and "replayed" and what-if scenarios easy to be built.
- 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 the set under control. The "scalability" requirement is closely related to this one.
- Groupware support: this is now an acknowledged fact that the 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 in 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, and the process is able to explain why these elements in the structural views, for example, have been chosen, which were discarded, and how this particular selection fulfils 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 in the development process as in post delivery analysis, these include architectural recovery and architectural conformance). The knowledge base formed by decisions is made more deep, or more decisions are included, or their possibilities for application increase.
- Dissemination and learning. The knowledge base 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 systems (in intellectual effort), 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… only if the border conditions (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 decision 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 the design decisions are: changes in the development team, design recovery needs, lost of designs, forward and backward trace between requirements and design products, etc. Therefore, an explicit representation of the design decisions becomes a key factor for building software architectures.

Design decisions must connect requirements and architectural products in order to 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 a 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.
- **Decision Rule**: This represents the name decision rule taken by the designer. 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 decisions. They are used to impose restrictions to a particular architectural element during the design construction process.
- **Associated Use Case**: They represent the number or name of one or more use cases associated to a particular design decision. This is used to connect the architectural product to requirements.

Figure 2 provides a graphical representation of a decision element which can be modelled employing a new UML element.

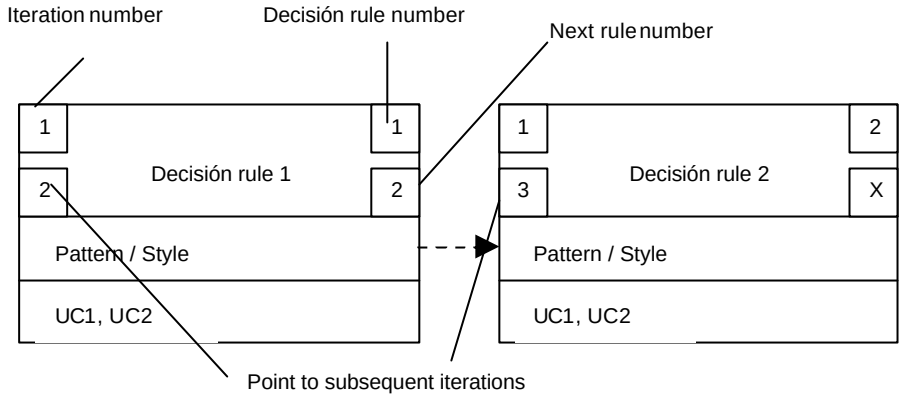

Fig. 2. Representation of the information included in the decision view of the software architecture

In this way, we can modify the figure proposed by Kruchten [6] to include the decision view as a intermediate element between requirements and other design views, such as figure 3 shows.

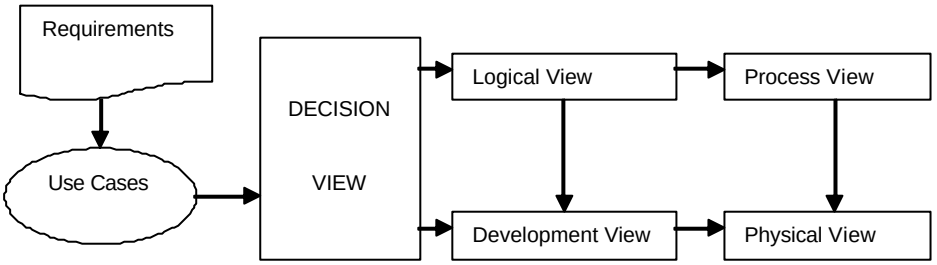

Fig. 3. The decision view model of software architecture

One key aspect when recording design decisions is how to associate these to architectural elements when we represent graphically these 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. For subsequent iterations, the design decisions elements will expand to describe the rationale of the design decisions taken during the process. Figure 4 shows an example of several iterations during the architecture construction process. The first iteration applies a layered style for the architecture and assigns a decision element for that. The following iterations apply 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.

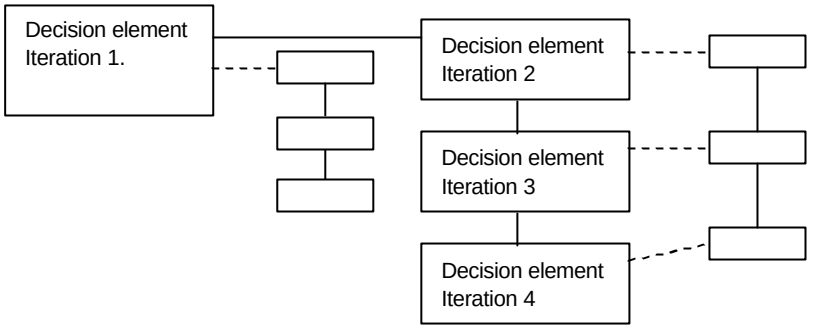

Fig. 4. Decisions elements associated to architectural elements for the several iterations in the architectural construction process

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. In fact, the authors understand this point as one of the key and deepest lessons to be taken from the open source communities and the social networks theories: the quality and size of the communication (links) are more important than the qualities of the pieces of information (the nodes). The practical application of this fact means that the decision view can be deployed as a hyperlinked documentation on top of the other architectural views: applying decisions is made by navigating that hyperlinked documentation, "building by browsing".

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

5. Katara M. and Katz S. Architectural Views of Aspects. Proceedings of AOSD 2003, Boston, USA, ACM, pp.1-10 (2003).
6. Kruchten P. Architectural Blueprints. The “4+1” View Model of Software Architecture. IEEE Software 12 (6), pp.42-50 (1995).
7. Shaw M. and Garlan D. Software Architecture, Prentice Hall (1996).
8. Alonso, A., León, G., Dueñas, J.C., de la Puente, J. A. Framework for documenting design decisions in product families development. Proceedings of the Third International Conference on Engineering of Complex Computer Systems", Como, Italia, 1997. September 1997. ISBN: 0-8186-8126-8.
9. Dueñas, J. C., Hauswirth, M. Hyper-linked Software Architectures for Concurrent Engineering. In Proceedings of Concurrent Engineering Europe 97, Erlangen-Nuremberg, Germany, 1997. pp: 3-10. Society for Computer Simulation. ISBN: 1-56555112-5.
10. 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 1998. Chapman and Hall, 1998. pp: 128-139. ISBN: 0-412-83530-4.
11. 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.
12. 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.
13. 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.
14. Jazayeri, M., Ran, A., van der Linden (eds) "Software Architecture for Product Families", Addison-Wesley, 2000. ISBN: 0-201-69967-2.