A Model to Represent Architectural Design Rationale

María Celeste Carignano
INGAR - CIDISI
UTN
Santa Fe, Argentina
celestec@santafe-conicet.gov.ar

Silvio Gonnet
INGAR - CIDISI
UTN - CONICET
Santa Fe, Argentina
sgonnet@santafe-conicet.gov.ar

Horacio Leone
INGAR - CIDISI
UTN - CONICET
Santa Fe, Argentina
hleone@santafe-conicet.gov.ar

***Abstract*—When developing a software system, its architecture must be considered so that it can be understood, updated, and improved. In general, considering the architectural artefacts is not enough. The reasons, assumptions and justifications bore in mind by the architects during the architecture design stage must be also known. Nevertheless, not all aspects analysed during the design process can be identified, especially all those alternatives that were evaluated and rejected. In the present contribution, a model to represent the rationale generated by architects during the architectural design is proposed so that it can last over time and it can be retrieved, analysed and reused whenever necessary. The model includes concepts representing architectural artefacts, reasons, assumptions, and decisions and reasoning elements status.**

Keywords- architecture, design rationale, design decision

I. INTRODUCTION

Bass et al. [1] define software architecture as a system structure or structures, which comprise software elements, externally visible properties of those elements, and relationships among them. The software architecture is the basis on which the fundamental guidelines for the system construction are defined.

The software architecture is really useful when it is understood by those who create, evaluate, use, query, reuse, and maintain it. This is achieved only if the rationale generated in its construction is captured and it is easily recoverable during its utilization. Also, it is necessary to know and understand the rationale generated in creating the software elements, relationships and involved properties. Burge et al. [2] define the rationale as the reasoning underlying the creation and use of artefacts.

Capturing the rationale generated in the architectural design allows stakeholders to answer the questions that arise along the software system life, from its conception to its death. Some of these questions are: (i) What happens if an architectural product or requirement is added, modified or deleted?; (ii) If a modification is made, do the architecture and the derived system remain valid?; (iii) What for and why were the current architectural products created?, Which reasons, justifications and considerations were taken into account?, Which architectural products allow architects to meet a requirement?; (iv) Which alternatives were studied at the moment of making decisions?; Why were the evaluated alternatives selected or rejected?, Do the reasons of rejection influence the new changes to be made?; (v) Can the system created from the analysed architecture give support to new requirements?; (vi) Do modifications derived from architecture maintenance impact on the instructions that are to be used in other process stages?; (vii) How was a given situation solved in other systems?, Which considerations that were taken into account in the design of other architectures are useful in designing the current system architecture?, Which are the existing solutions to a specific architectural problem?, Which ones were used in previous architectures?.

All non-documented rationale is missed or ignored over time. This makes it impossible to obtain conclusions from the given questions.

Nowadays, there are various tools that implement different architectural description languages (ADL), for example ACME, Rapide, among others. These do not allow designers to specify the applied rationale to reach a final architecture. There are design assistants, such as ArchE, which provide support to the design process through reasoning frameworks, but they do not allow architects to document the rationale generated beyond the tactics to be applied. The decisions taken by the architects are considered to be based on their own experiences, although these are not formalized in a reasoning framework. These can be influenced by the software construction company policies, by consistency with specific expected solutions, by the knowledge or skills of the team members that can be exploited in order to obtain results faster or better, etc. There are various contributions about knowledge management used in architectural design ([3], [4], [5], [6], [7] among others). Some frameworks or tools were created according to these works, for example Archium [8], PAKME [5] and AREL [7]. Each contribution adopts a particular approach that provides some answers to the previously mentioned questions.

In order to answer these questions, this paper proposes a model to represent the rationale generated during the architectural design definition stage.

II. PROPOSED MODEL

The proposed model was specified using UML 2.0. The advantages of proposing the model using UML on the use of templates [9] are: better information organization so that it can be easily retrievable, reduction of information redundancy, and avoiding the use of natural language to indicate relationships among concepts. This approach enables us to represent how and why an architectural product, or some part of it, is designed the way it is [4].

Fig. 1 represents the fact that a software system is developed in order to satisfy the stakeholders' needs (*Stakeholder* in Fig. 1). These needs can be associated to the specific functionalities that the system must provide (*FunctionalRequirement* in Fig. 1) or to the characteristics that the system or a functionality should have (*NonFunctionalRequirement* in Fig. 1).

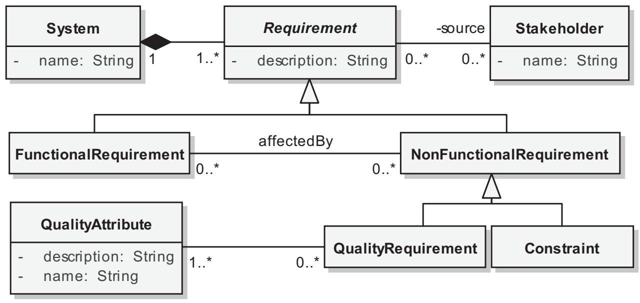

Figure 1. Description of the stakeholders' needs through requirements.

Non-functional requirements may specify that the software system (or one of its functionalities) should achieve a given quality attribute (*QualityRequirement* associated to *QualityAttribute* in Fig. 1); or they can point out the need of fulfilling a constraint imposed by the stakeholders (*Constraint* in Fig. 1). Some examples of quality attributes are: response time, security and performance. An example of constraint is: the demand for using a specific technology for the system development.

During the software architecture definition, some architectures that meet users' requirements are created using different resolution approaches. Each solution is considered as an alternative (*ResolutionAlternative* in Fig. 2) that can be left aside, selected or rejected (*ResolutionAlternativeState* in Fig. 2). Thus, the architect can work with various parallel solution alternatives to finally determine which one is the most promising one to be continued. This happens, for example, when the technology to be used is not selected yet and it is necessary to specify which is the most suitable one to satisfy the requirements. For that reason, various teams of architects suggest the alternative architectures resulting from using each technology and then they evaluate which is the best one.

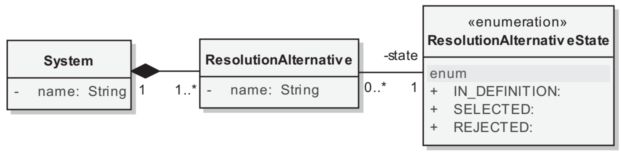

Figure 2. Resolution alternatives.

A solution alternative is a set of architectural design decisions. Before defining an architectural design decision, architectural case and architectural solution concepts are introduced. An architectural case (*ArchitecturalCase* in Fig. 3) is a problem or situation that needs an answer from the architectural point of view. Some examples of architectural cases are: preventing unauthorized users from accessing the application, a specific functionality or their data; making modifications to the system in the shortest time and with as few resources as possible; designing the elements responsible for executing one or more functionalities, among others. Architectural solutions (*ArchitecturalSolution* in Fig. 3) are generic solutions to architectural cases, which propose design fragment types [10] (*DesignFragmentType* in Fig. 3) so that they can be applied to solve the case. Architectural solutions can be formal solutions such as architectural tactics; or descriptions of solutions arose from the architects' experiences.

The *ArchitecturalCase* class is useful for the specification of common problems or situations that arise during the definition of an architecture. The *ArchitecturalSolution* class represents the generic solutions that can be applied in order to share and reuse the captured knowledge. Architectural solutions and architectural cases are shared by all the architectures created from the proposed model, so they can feed each other and the knowledge obtained in the design of new architectures can be used.

When system requirements are analysed, one or more architectural cases may be identified. Users' requirements, both functional and non-functional, are documented by creating instances of *Requirement* subclass. Then, the architectural cases are identified into the requirement context through the *Scenario* class (Fig. 3). For example, given the requirement: "the system must record clients' orders"; a possible architectural case is: "design the elements responsible for executing one or more functionalities". This association is carried out by a *Scenario*.

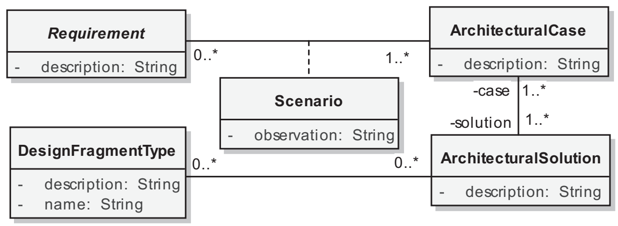

Figure 3. Architectural Cases and Solutions.

Scenarios are fulfilled by architectural design decisions (*ArchitecturalDesignDecision* in Fig. 4). As it was introduced, a set of architectural design decisions compounds a resolution alternative (Fig. 4). Decisions are materialised by the creation or selection of architectural design products and documenting the appropriated details. This concept can be spread by adding the attributes identified by Krutchen [11], such as: epitome, author, time-stamp, and history.

The architectural design products are obtained as a result of the selection of an architectural solution to solve part or the whole scenario associated to each architectural design decision (through *SelectedArchitecturalSolution* in Fig. 4). The three more important types of products are: (i) design fragments (*DesignFragment* in Fig. 5), such as objects or architectural styles [10]; (ii) properties of the existing design fragments (*Property* in Fig. 5); and (iii) documentation elements (*DocumentationElement* in Fig. 5) that indicate information that must be taken into account in other construction software stages (represented by the *destination* relationship with *Activity* class in Fig. 5). The *Activity* class is consistent with the *Activity* class of the *ProcessStructure* package of the SPEM metamodel [12], which allows modellers to represent several software processes, such as RUP. *DocumentationElement* allows designers to document all information considered and specified during the architectural design and that has influence on other stages, such as during detailed design, implementation or tests. Thus, it avoids depending on the architects' memory in order to take into account the considerations detected at the right moment. By using this documentation element, it is not intended to document rationale but definitions made during the architectural design and cannot be captured in a property or design fragment.

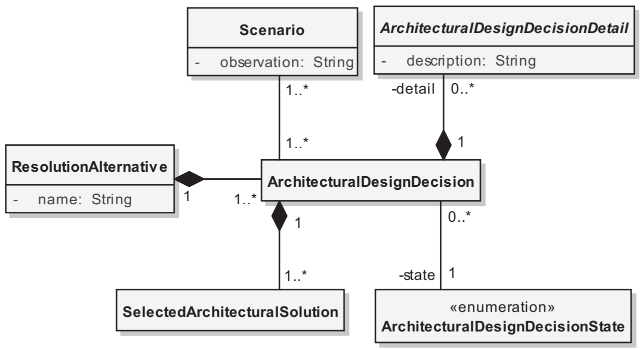
Scenario
- observation: String
ArchitecturalDesignDecisionDetail
- description: String
1..*
-detail 0..*
1..*
1
ResolutionAlternative
- name: String
1
1..*
ArchitecturalDesignDecision
1
0..*
1..*
-state 1
SelectedArchitecturalSolution
«enumeration» ArchitecturalDesignDecisionState

Figure 4. Architectural Design Decision.

When the architect is working on the software architecture solution alternative, various decisions are taken. Each one has a status in order to identify its degree of acceptance (*ArchitecturalDesignDecisionState* in Fig. 4). These statuses might be, for example: idea, tentative, decided, approved, challenged, rejected, and obsolesced [11].

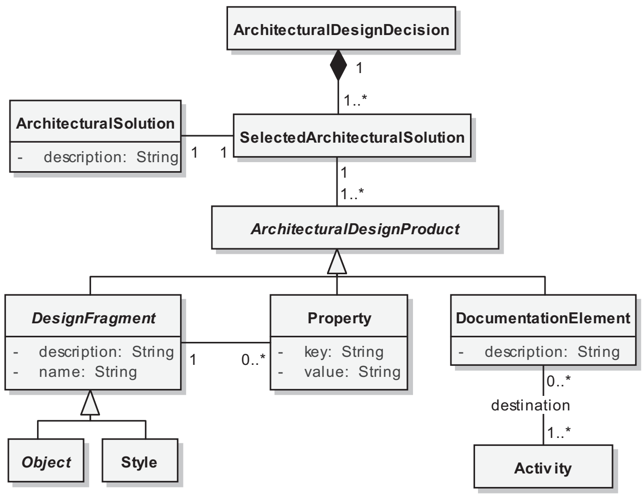

Figure 5. Architectural design products.

Similarly, the approach presented by Tang et al. [7] allows architects to work with the concept of Alternative Architecture Rationale associated to particular decisions, and the approach presented by Akerman and Tyree [13] allows designers to work with the concept of Alternative associated to a particular architecture decision, but our proposal also allows architects to work on solution alternatives of the complete architecture as it was presented previously.

When an architectural design decision is taken, the profits and drawbacks of its application must be analysed and evaluated. When this information is not trivial, it can be documented (*ArchitecturalDesignDecisionDetail* in Fig. 6).

So, the reasoning made by the architects while working in the design decision, either rejected or selected, is available at the moment of using and consulting the architecture.

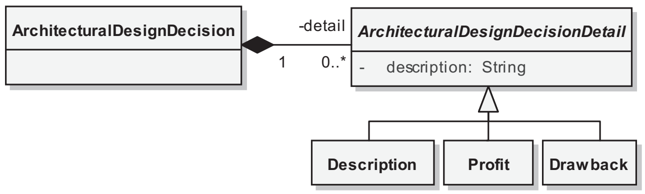
ArchitecturalDesignDecision
-detail
1
0..*
ArchitecturalDesignDecisionDetail
- description: String
Description
Profit
Drawback

Figure 6. Architectural design decisions details.

A design decision may need the creation of new requirements. Fig. 7 represents a *Requirement* as a product of an architectural design decision. This fact is modelled as an association class (*GenerationDetail* in Fig. 7) between *Requirement* and *ArchitecturalDesignDecision*. For example, when an architect decides to use a passwords scheme to solve a security quality requirement, new instances of *FunctionalRequirement* class are created, which have the purpose of manage the user's login and logout of the system using their private password. These new requirements must be considered only into the resolution alternative in which they were created.

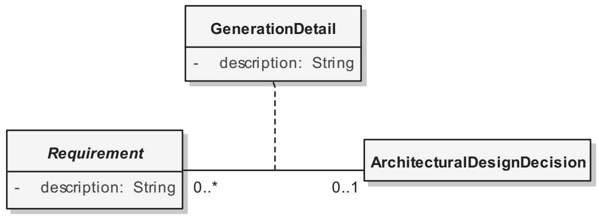

Figure 7. New requirements created by a decision taken.

III. CASE STUDY

This brief case study presents how the proposed model captures the architectural design of an orders record system through mobile devices. Its main functionalities were: (i) the seller must record the clients' orders through mobile devices; and (ii) the seller must synchronize offline the data of orders, products and clients with the central server. From the last requirement, the architect identifies the architectural case: "Synchronize the data offline from mobile device with the data from the central server". This architectural case was linked with the requirement by a concrete scenario and with the architectural solution: "Create synchronisation elements, one into the server and one into the mobile device, which responsibilities will be to verify the change in its environment and inform the changes between each other". Therefore, this architectural solution was applied (selected) during the architectural design decision taken in order to meet that scenario.

The argued details were (instances of *ArchitecturalDesignDecisionDetail*):

- Profits: "When using different synchronization elements, the central server may become independent in a way in which the information in the mobile devices is storage".
- Drawback: "New data to transfer implies more changes (in both synchronizers)".

The architectural design products generated as result of this architectural design decision were: the *MobileSynchronizer* component (it was allocated to the mobile device) and the *ServerSynchronizer* component (it was allocated to the central server). These new components are presented in Fig. 8.

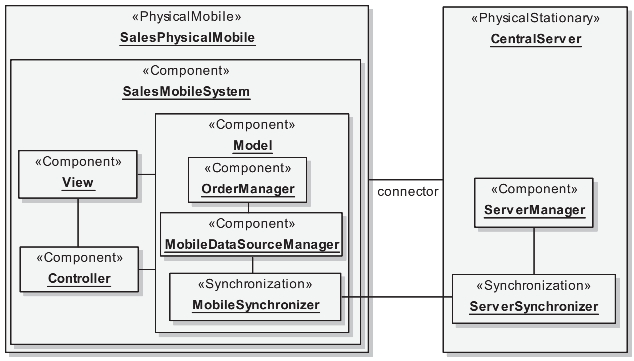

Figure 8. Architecture after design decision.

This simple example illustrates how the model was used to represent the rationale made by the architects when they design the architecture of each system considered. Finally, some situations requiring retrieving the documented rationale were considered. These situations were related to: (i) new requirement request; (ii) knowledge transfer from experts to new architects; (iii) knowledge to support maintenance activities; (iv) architecture interpretation for training new members; and (v) reuse of a successfully applied solution. Thus, the proposed model could document the rationale employed by the architects in the architectural design of the proposed systems. Then, this rational can be retrieved and used in other situations that otherwise could have not been solved.

IV. CONCLUSIONS

We present a model that allows architects to represent the rationale generated during software system architecture design.

This model was used in order to document the rationale employed in architectural design of several software systems. During the architectural design of each system, many decisions were taken and their associated rationales were documented, specifying decision details and process design products. *AchitecturalCase* and *ArchitecturalSolution* previously defined were used in order to identify the architectural cases detected in all systems and their possible solutions.

As a result, a set of architectural products (objects, styles, properties and documentation elements) and the details of the rationale associated with their definitions were obtained for each architecture.

In future works, the rationale captured with this model will be used in order to facilitate the creation of new software architectures based on past experiences. As a consequence, some artificial intelligence techniques should be applied in order to learn from the captured rationale.

ACKNOWLEDGMENT

The authors acknowledge financial support received from CONICET, Universidad Tecnológica Nacional and Agencia Nacional de Promoción Científica y Tecnológica (PICT 34602, IP-PRH 2007).

REFERENCES

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

[2] J. Burge, J. Carroll, R. McCall, and I. Mistrík. Rationale-Based Software Engineering. Springer-Verlag, 2008.

[3] J. S. van der Ven, A. Jansen, J. Nijhuis, and J. Bosch. Design Decisions: The Bridge between Rationale and Architecture, chapter 16, Rationale Management in Software Engineering, A. Dutoit, R. McCall, I. Mistrkik, and B. Paech (Eds.), pp. 329-346, Springer-Verlag, April 2006.

[4] M. Babar, I. Gorton, and B. Kitchenham. A Framework for Supporting Architecture Knowledge and Rationale Management, chapter 11, Rationale Management in Software Engineering, A. Dutoit, R. McCall, I. Mistrkik, and B. Paech (Eds.), pp. 237-254, Springer-Verlag, April 2006.

[5] M. A. Babar, I. Gorton. A Tool for Managing Software Architecture Knowledge. Second Workshop on SHAring and Reusing architectural, Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI'07), 2007.

[6] L. Bass, P. Clements, R. Nord, J. Stafford. Capturing and Using Rationale for a Software Architecture, chapter 12, Rationale Management in Software Engineering, A. Dutoit, R. McCall, I. Mistrkik, and B. Paech (Eds), pp. 255-272, Springer-Verlag, April 2006.

[7] A. Tang, Y. Jin, J. Han. A rationale-based architecture model for design traceability and reasoning. The Journal of Systems and Software 80, pp. 918-934. Elseiver, 2007.

[8] A. Jasen, J. van der Ven, P. Avgeriou, D. Hammer. Tool support for Architectural Decisions. Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture (WICSA'07), pp. 44-53, 2007.

[9] A. Akerman, J. Tyree. Architecture Decisions: Demystifying Architecture. IEEE Software, 22(2), pp. 19–27, 2005.

[10] M. C. Carignano, H. Leone, S. Gonnet. Modelo de Soporte de Decisiones de Diseño Arquitectónicas. XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 390-399, 2008.

[11] P. Krutchen. An Ontology of Architectural Design Decisions in Software-Intensive Systems. 2nd Groningen Workshop Software Variability, pp. 54-61, 2004.

[12] Software & Systems Process Engineering Meta-Model Specification v. 2.0, Object Management Group Specification (2008), http://www.omg.org

[13] A. Akerman, J. Tyree. Using ontology to support development of software architectures. IBM Systems Journal, Vol 45, No 4, pp. 813-825, 2006.