Modelling architectural decisions under changing requirements

Marcin Szlenk, Andrzej Zalewski, Szymon Kijas
Institute of Control and Computation Engineering
Warsaw Univeristy of Technology
Warsaw, Poland
{*m.szlenk,a.zalewski,s.kijas*}*@elka.pw.edu.pl*

***Abstract*—One of the approaches for documenting software architecture is to treat it as a set of architectural design decisions. Such decisions are always made in the context of requirements that must be fulfilled and in the context of decisions that were made before. Currently, models for representing architectural decisions are mainly concentrated on showing the decision making process of the initial architectural design. However, decisions that have been made in such a process may need to be changed during further evolution and maintenance of the software architecture, typically in response to the new or changed requirements.**

A graphical modelling notation for documenting architectural decisions (Maps of Architectural Decisions) has been developed by our team. In this paper, it is presented how this notation could be used to model architectural decisions under changing requirements. It is proposed how one decision change could be effectively propagated through the rest of the architectural decision model and how a rigorous and tool-supported process of updating such models could be organized.

***Keywords*-architectural decisions; graphical models; requirement changes;**

I. Introduction

Architectural decisions [2], [11] are often perceived as another wave in architecture modelling that deals with the representation, capture, management, and documentation of the design decisions made during architecting [8]. As all other decisions in the software development process these also may undergo changes, e.g in one of the succeeding iteration steps in the iterative development process or during the software maintenance phase. Such alterations are usually related to changes in the requirements: a new requirement has appeared or has been taken into account what forces a software architect to change one or more decisions. Architectural decisions are often related with each other and changing one of them may affect the more extensive part of the decision model. Performing the changes in the architectural decision models in a rigorous way should be considered an important issue and this is the problem this paper addresses.

II. Related Work and Motivation

Architectural decisions are usually represented as text records [1], [4], [11], sometimes accompanied with illustrating diagrams [3]. The limitations of the textual models are well-known in the genre of software engineering. Therefore, sets of hundreds of architectural decisions necessary to represent sufficiently architecture of a large system, are difficult to comprehend, analyse, verify and ensure completeness and consistency or even just to navigate through them. For that reason, diagrammatic (based on graphs) models have been proposed as a way to represent architectural decision and decision making process (see [10], [12], [13]) in a more comprehensive way. Graphical models and tools supporting architectural decision and decision-making have been presented in [5], [10], [12].

Decision classifications have been developed to help to organise large sets of architectural decisions. Most influential classifications by Kruchten [7] (*existence*, *non-existence*, *property* and *executive* decisions) and Zimmermann [13] (*executive*, *conceptual*, *technology* and *vendor asset* decisions) substantially help to navigate through a set of architectural decisions. However, these categories are not always precise, and in many cases can confuse engineers. In both references, not only the categories of architectural decisions have been defined but also the possible kinds of relations between such decisions have been determined. In [7] Kruchten indicates ten different kinds of relations between architectural decisions.

Another architectural decision model and diagrammatic notation, called "Maps of Architectural Decisions" (MAD), have been developed by our team and presented in [12]. MAD has been created to support architect-practitioners working on systems evolution. It does not impose any predefined classification or hierarchy of architectural decisions and assumes a limited number of relation kinds between architectural decisions. This makes the model of the decision process intuitive and easy to comprehend. To explain the choices made and capture their rationale, the entire decision situation is presented, including: the decision topic (or problem), considered design options, relevant requirements, the advantages and disadvantages of every considered option.

Although there are many approaches for representing and capturing architectural decisions, it seems that in all cases the following scenario is assumed: one has an initial set of requirements and according to these requirements the architectural decisions are made and documented. However, an important question arises: what activities should be done

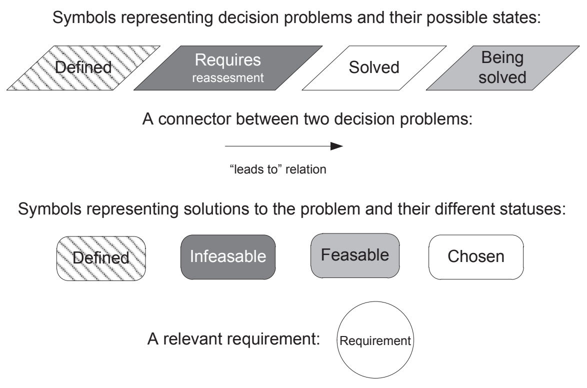
Symbols representing decision problems and their possible states:
Defined
Requires reassesment
Solved
Being solved
A connector between two decision problems:
"leads to" relation
Symbols representing solutions to the problem and their different statuses:
Defined
Infeasable
Feasable
Chosen
A relevant requirement:
Requirement

Figure 1. The part of the MAD notation

when the initial set of requirements changes but some or all of the decisions have been already captured in the model? Such a situation is quite usual during a project with iterative development [6] and typical for the maintenance phase when requirements for the next release of a system appear. It seems that, so far, this problem has not been given much attention in the related works on architectural decision modelling. Thus, we would like to address it here. It appears that the MAD notation offers some constructions particularly useful for this problem, as it will be shown further in the paper.

III. Maps of Architectural Decisions

MAD was crafted to assist architects in architecture decision-making, without enforcing any particular architecting approach or decision-making order. MAD works similarly to popular mind maps used to present a problem structure graphically. The MAD models are built up of the following elements:

- *Decision problem* – represents the architectural issue being considered;
- *Connector* – in its basic form shows that one solved problem led an architect to the one indicated by an arrow (a "leads to" relation);
- *Solution* – represents a single solution to the architectural problem considered;
- *Requirement* – represents a requirement relevant to a given architectural problem;
- *Decision-maker* - represents a person or a group of people responsible for the resolution of a related architectural problem;
- *Pro or Con* - represents a single advantage or disadvantage of a given solution.

These elements have additional attributes, e.g. name, description, state, creation date and resolution date for the decision problem. For the detailed description please refer to [12]. The most important elements of the notation are shown in Fig. 1.

IV. Decision Problem Life Cycle

MAD concentrates on showing a decision making process and its progress. It does not provide classification categories for architectural decisions but instead introduces the concept of a decision problem's state, proposing four possibilities: *defined*, *being solved*, *solved*, and *requires reassessment* (see Fig. 1). This distinguishes MAD from the other approaches mentioned in Sec. II. However, in [12], the meaning of these states have been explained only superficially and no state transition rules have been defined. To define such rules in this paper, first the concept of an architectural decision problem's context will be introduced.

The context, in which the architectural decision problem is being considered, contains both the requirements and the decisions that have been already made [2]. To be more precise, not all the requirements and decisions made before should be considered here but, naturally, only these which are relevant to the given decision problem. In the MAD model the requirements are directly attached to decision problems they are relevant to, and the earlier decisions (chosen solutions), that led to the given problem, can be easily discovered by tracing the "leads to" relationship. Let us first introduce two definitions:

Definition 1 (Simplified MAD model)**.** By a simplified MAD model we understand a tuple $(Problems, leadsTo, requirements, solution)$, where:

- $Problems$ is a set of decision problems,
- $leadsTo \subseteq Problems \times Problems$ is a set of pairs of decision problems connected through the "leads to" relation,
- $requirements(p)$ is a set of requirements relevant to the problem p, and
- $solution(p)$ is a single-element set containing a finally selected solution to the problem p (if the solution is not selected yet, then $solution(p)$ is undefined).

Definition 2 (Reachability relation)**.** Let M be a simplified MAD model and a relation $\succ \subseteq Problems \times Problems$ be the transitive closure of the relation $leadsTo$. The relation $\succ$ will be called a reachability relation for the model M. If $p \succ q$ then we will say that the problem q is reachable from the problem p.

According to the informal definition mentioned earlier, the context of a problem can be now defined as shown below.

Definition 3 (Context)**.** Let M be a simplified MAD model, $\succ$ be the reachability relation for the model M. The context of a problem $p \in Problems$ in M is defined as:

$$context(p) = requirements(p) \cup \bigcup_{q \succ p} solution(q).$$

Let us now discuss the life cycle of a decision problem in the MAD model. When a new decision problem is added

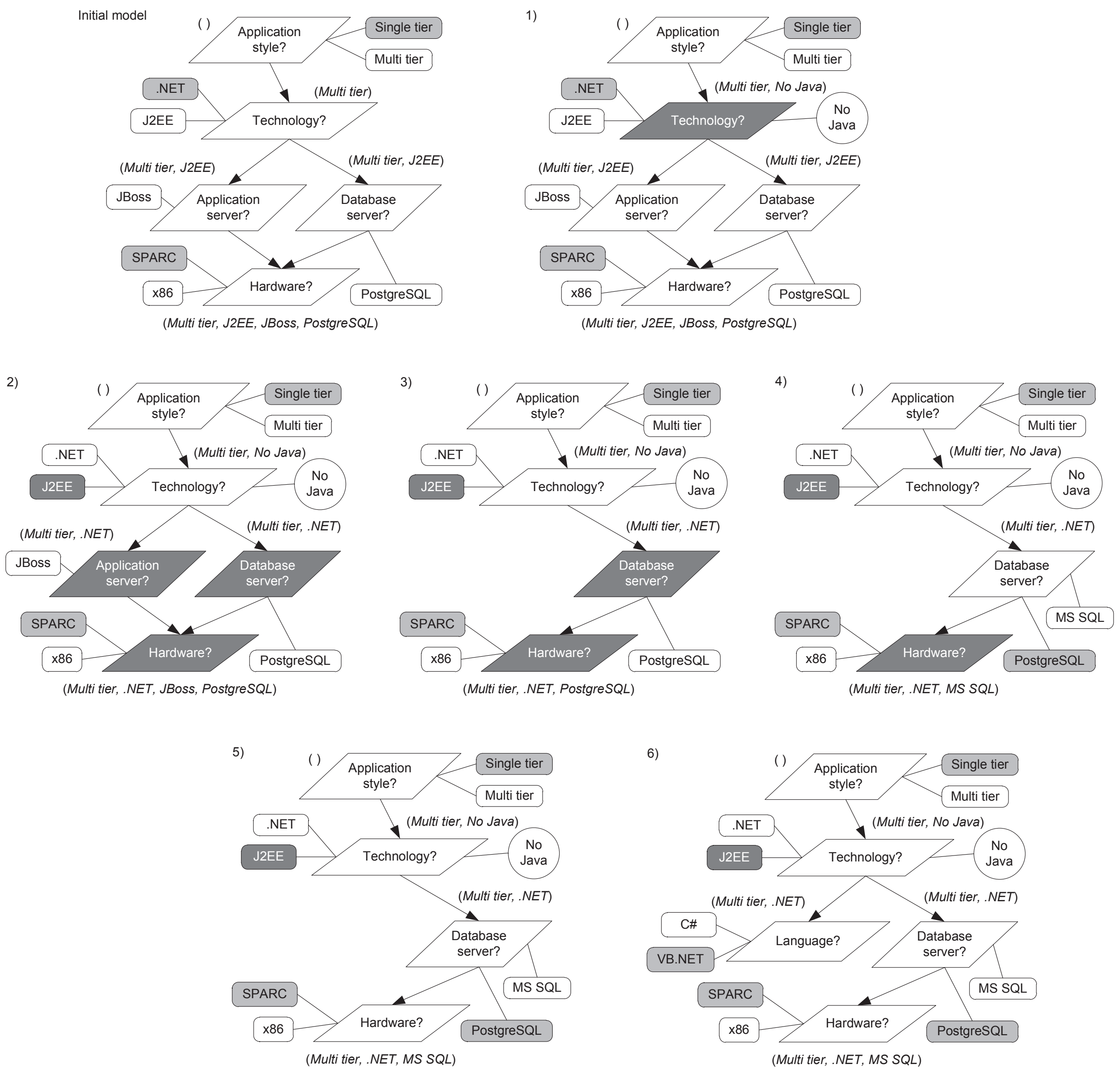

Figure 2. An example of model rebuilding

to the model it is in the "defined" state. Next, when the possible solutions are being considered the problem is "being solved", and once one of the solutions connected to the problem becomes "chosen", the problem becomes "solved" and can lead to new problems. The above state transitions are quite intuitive, however, the question remains: when should a given problem change its state into "requires reassessment"? We argue, that such a change should always be motivated by the alteration of the problem's context, i.e. the problem should change its state into "requires reassessment" if and only if its context has changed. The information about the problem's context is captured in the MAD model, and thus, such a state transition should occur automatically in response to certain changes in the model. When the context of the decision problem has changed due to the changes of the previous decisions, the problem itself may not occur any more and needs to be removed from the model (what will take place in the example shown in Sec. V).

V. An Example of Model Rebuilding

When a new requirement appears, the software architect can decide whether it is relevant to one or more of the decisions captured in the architectural decision model. In the MAD model this new requirement will be then connected to proper decisions, changing at the same time their contexts and their status into "requires reassessment" as a result. This would initiate the process of model rebuilding. An example of such a process is presented in Fig. 2 and explained below.

The initial MAD model shown in Fig. 2, although academic in nature, should be understandable to the reader without difficulty. According to Zimmermann's classification [13], the model combines both conceptual, technology, and vendor asset level decisions. The current context of each decision problem is shown in the parentheses. This is for informative purpose only and it is not a standard element of the MAD notation. For the simplicity, only the newly added requirement has been shown in the model. The process encompasses six steps:

1) The new requirement (a request for a change) telling that our company is moving from the Java technology has been connected with the "Technology?" decision problem.
2) The architect has chosen a different technology as a solution to the "Technology?" problem.
3) In the case of .NET technology there is no such concept as an application server [9], so the "Application server?" problem does not occur any more and has been removed from the model.
4) An alternative solution for the "Database server?" problem has been added and chosen.
5) The "Hardware?" problem has been analyzed in the new context, but the solution remained unchanged.
6) Because there is more than one .NET programming language, the new solution for the "Technology?" problem has generated a new decision problem: "Language?". Two solutions have been considered here and one of them has been chosen.

Please note, the MAD model does not have to be completed (i.e. all the decision problems are solved) to add new requirements to it, and perform the updating process similar to the one presented above.

VI. Conclusions and Further Work

MAD models have been developed to assist system architects in a similar way as mind maps do. This helps to capture architectural knowledge as it gradually comes to light while elaborating the architecture (i.e. decision-making). In this paper it has been shown how MAD models can be built in an iterative way, where new requirements appear after the whole or parts of the model were created. It is a natural continuation of our previous work ([12]), as MAD has been motivated by the rapid, random changes typical for the evolution of systems supporting emergent organisations.

MAD has been validated in the real life conditions of one of the largest telecom firms in Poland and a software tool supporting MAD has been also developed [12]. Future work will be focused around extending this tool to support the process of rebuilding models in response to the new requirements. This support will be based on the idea presented in this paper, leading an architect through interactively guided and monitored steps. Next, further empirical evaluation of the concept will be conducted.

Acknowledgment

This work was sponsored by the Polish Ministry of Science and Higher Education under grant number 5321/B/T02/2010/39.

References

[1] M. Ali Babar, T. Dingsøyr, P. Lago, and H. van Vliet, Eds., Software Architecture Knowledge Management: Theory and Practice, Springer-Verlag, 2009.

[2] J. Bosch and A. Jansen, "Software Architecture as a Set of Architectural Design Decisions," Proc. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA05), IEEE Computer Society, pp. 109–120, 2005.

[3] R. Capilla, F. Nava, and J. C. Dueñas, "Modeling and Documenting the Evolution of Architectural Design Decisions," Proc. of the Second Workshop on Sharing and Reusing Architectural Knowledge Architecture, Rationale, and Design Intent, IEEE CS Press, pp. 9–16, 2007.

[4] N. B. Harrison, P. Avgeriou, and U. Zdun, "Using Patterns to Capture Architectural Decisions," IEEE Software, vol. 24, no. 4, pp. 38–45, 2007.

[5] A. Jansen, P. Avgeriou, and J. S. van der Ven, "Enriching Software Architecture Documentation," Journal of Systems and Software, vol. 82, no. 8, pp. 1232–1248, 2009.

[6] P. Kruchten, The Rational Unified Process – An Introduction, 3rd ed., Addison-Wesley, 2003.

[7] P. Kruchten, P. Lago, and H. van Vliet, "Building Up and Reasoning About Architectural Knowledge" in C. Hofmeister (Ed.), Proceedings of Second International Conference on the Quality of Software Architectures (QoSA 2006), LNCS 4214, pp. 43–58, Springer-Verlag, 2006.

[8] P. Kruchten, R. Capilla, and J. C. Dueñas, "The Role of a Decision View in Software Architecture Practice," IEEE Software, vol. 26, no. 2, March/April 2009, pp. 36–42.

[9] Microsoft Application Architecture Guide (Patterns & Practices), 2nd ed., Microsoft Press, 2009.

[10] M. Shahin, P. Liang, and M. R. Khayyambashi, "Improving understandability of architecture design through visualization of architectural design decision," Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge, ACM, pp. 88–95, 2010.

[11] J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture," IEEE Software, vol. 22, no. 2, March/April 2005, pp. 19–27.

[12] A. Zalewski, S. Kijas, and D. Sokołowska, "Capturing Architecture Evolution with Maps of Architectural Decisions 2.0," in I. Crnkovic, V. Gruhn, and M. Book (Eds.), ECSA 2011, LNCS 6903, pp. 83-96, Springer-Verlag, 2011.

[13] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster, "Managing architectural decision models with dependency relations, integrity constraints, and production rules, " Journal of Systems and Software, vol. 82, no. 8, pp. 1249–1267, 2009.