Towards Precise Architectural Decision Models

Marcin Szlenk[1]

[1] Warsaw University of Technology, Institute of Control and Computation Engineering, Nowowiejska 15/19,00-665 Warsaw, Poland, m.szlenk@elka.pw.edu.pl

Abstract. One of the modern approaches for documenting software architecture is to show the architectural design decisions that led an architect to the final form of software architecture. However, decisions that have been made in such a process may need to be changed during further evolution and maintenance of the software architecture. The main reasons for these changes are new or changed requirements. In our team we have developed a graphical modelling notation for documenting architectural decisions, called Maps of Architectural Decisions, that can support the process of making changes in the software architecture. In this work we define a formal background for the controlled process of making changes in architectural decision models that are documented using that notation.

1 Introduction

Software architecture is developed as a result of numerous interrelated decisions. The architecture itself, these decisions and the context of these decisions create the architectural knowledge. Documenting architectural decisions is a new wave in architecture modelling [2, 12]. It deals with the representation, capture, management, and documentation of the design decisions made during architecting [9]. In the process of software maintenance and evolution, architectural decisions may undergo changes in response to new or changed requirements. Such 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 is an important problem we want to address in this work. The proposed solution is based on the modelling notation called Maps of Architectural Decisions (MAD) [13]. This work extends our previous work [11] mainly by introducing the concept of decision consistency (Sect. 5) and defining the formal metamodel of the MAD notation (Sect. 6).

2 Related Work and Motivation

A typical representation of architectural decisions are text records [1, 4, 12], that are sometimes accompanied with illustrating diagrams [3]. Many diagrammatic (based on graphs) models have been also proposed as a way to represent architectural decision and decision making process in a more comprehensive way (see [10, 13, 14]). Graphical models and tools supporting architectural decisions and decision-making have been presented in [6, 10, 13].

Decision classifications have been developed to help to organise large sets of architectural decisions. Most influential classifications by Kruchten [8] (*existence*, *non-existence*, *property* and *executive* decisions) and Zimmermann [14] (*executive*, *conceptual*, *technology* and *vendor asset* decisions) substantially help to navigate through a set of architectural decisions. 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 [8], Kruchten indicates as much as ten different kinds of relations between architectural decisions.

Another architectural decision model and diagrammatic notation (MAD) have been developed by our team and presented in [13]. 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 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 [7] and typical for the maintenance phase when requirements for the next release of a system appear.

The new or changed requirements will usually lead to a changed decision model, but the main problem here is how to perform these changes in a controlled way and verify whether each change is justified by the context. The precise formal definitions would be welcomed here, as they open the further possibility of automatic verification. 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 in terms of models expressed in the MAD notation.

3 Maps of Architectural Decisions

MAD notation works similarly to 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 [13]. The most important elements of the notation are shown in Fig. 1.

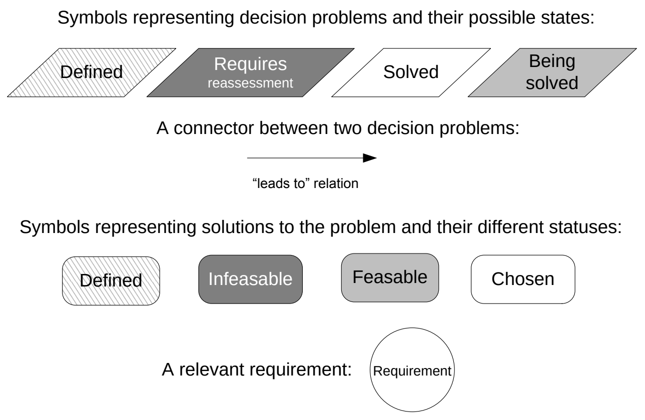
Symbols representing decision problems and their possible states:
Defined
Requires reassessment
Solved
Being solved
A connector between two decision problems:
"leads to" relation
Symbols representing solutions to the problem and their different statuses:
Defined
Infeasable
Feasable
Chosen
A relevant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

Fig. 1 The MAD notation

3.1 Decision Problem Life Cycle

The main objective of MAD is to show a decision making process and its progress. It introduces the concept of a decision problem's state, proposing four possibilities: *defined*, *being solved*, *solved*, and *requires reassessment* (see Fig. 1). The state transition rules have been defined for MAD using the concept of a decision problem's context. 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. In [11] the three definitions have been introduced:

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.

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:

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

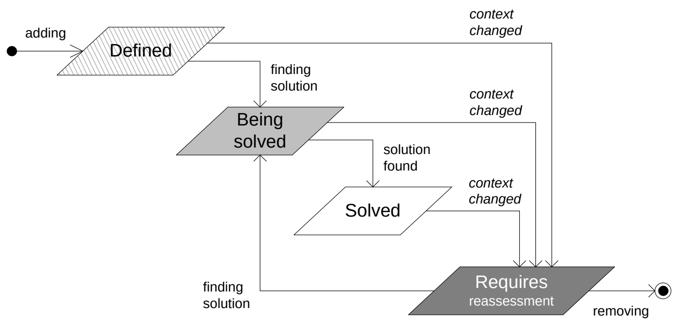

Fig. 2 The decision problem life cycle

Let us now discuss the life cycle of a decision problem in the MAD model. When a new decision problem is added 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. Whenever the problem's context is changing, the problem should automatically change its state into "requires reassessment". 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 in such a situation it should be removed from the model. The described decision problem life cycle is summarized in Fig. 2.

3.2 *Model Rebuilding*

The decision problem life cycle leads us to the rigorous process of making changes in MAD models in response to new requirements. When the new requirement appears, the software architect can decide whether it is relevant to one or more of the decisions captured in the MAD 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. The similar situation will occur when one of the requirements already existing in the model is changing.

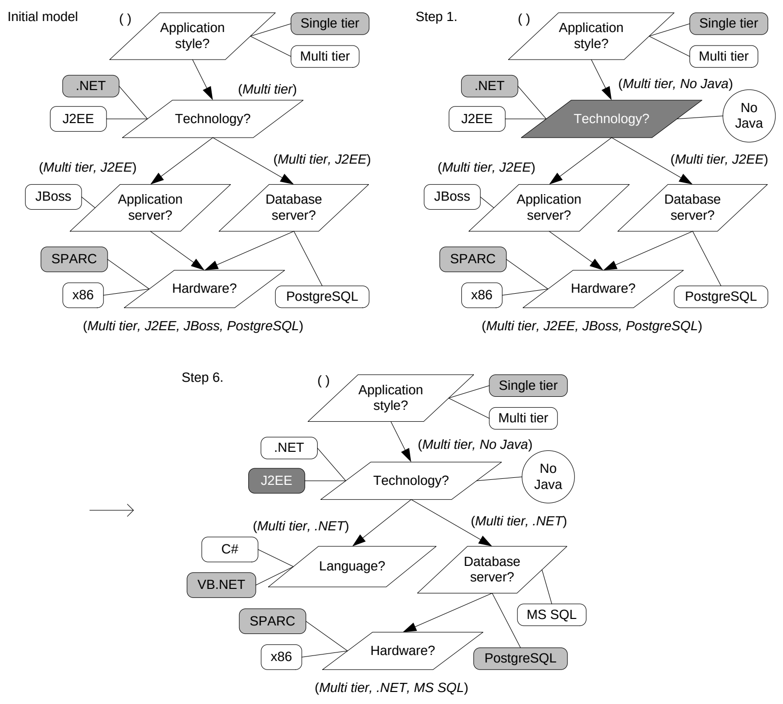

Fig. 3 Model rebuilding

Changing the status of any problem into “requires reassessment” may initiate the process of model rebuilding. An example of such a process has been presented in [11]. That process encompasses six steps. In Fig. 3, we present only the first step and the final result of the whole rebuilding process. The intermediate steps have been omitted. In the first step the new requirement telling that our company is moving from Java technology has been connected with the “Technology?” decision problem. For detailed description of the successive steps, please refer to [11].

5 Decision Consistency in MAD Models

As it has been shown in the previous section, the appearance of new requirements may result in a changed architectural decision model. Thus, the evolution process of a system may result in a sequence of architectural decision models, where two consecutive models reflect the single step of system evolution. Let us consider such a sequence of two architectural models (i.e. one evolution step) and call them M_1 and M_2. Let p be one of the solved decision problems (a problem in the "solved" state) in the model M_1. There are two possibilities:

- the problem p does not occur any more and is not present in M_2,
- the problem p does still occur and is present in both the models: M_1 and M_2.

In the second case, if the problem p is also solved in the model M_2 then:

- the problem p has the same finally selected solution (a solution in the "chosen" state) in both of the models, or
- the problem p has a different finally selected solution, but in that case the contexts of p in M_1 and M_2 must be also different, otherwise there would not be any justification for changing a solution to the problem p (i.e. in the meantime p must have been in the "requires reassessment" state).

Let us now define the above relation between two MAD models formally.

Definition 4 (*Decision consistency relation*)
Let M_1 and M_2 be two simplified MAD models. We say that the decisions in M_1 and M_2 are *consistent* if they have the same finally selected solution or they contexts are different:

$$\begin{gathered}\forall p \in \mathit{Problems}_1 \cap \mathit{Problems}_2 \cdot \\ \mathit{solution}_1(p)\ \textbf{is defined} \wedge \mathit{solution}_2(p)\ \textbf{is defined} \Rightarrow \\ \mathit{solution}_1(p) = \mathit{solution}_2(p) \vee \mathit{context}_1(p) \neq \mathit{context}_2(p).\end{gathered}$$

If M_2 is a result of rebuilding M_1 in a process of architecture evolution, the decisions in the model M_1 must be consistent with the decisions in the model M_2 in terms of the above definition. From the definition of the decision consistency relation it can be easily seen that this relation is both symmetric and reflexive.

6 Metamodel of MAD Notation

Although MAD notation offers constructions useful not only for documenting architectural decisions but also for making rigorous changes in a set of interrelating decisions, its biggest drawback when it comes to automatic verification of changes is its lack of precise metamodel. In this section, the metamodel of the part of MAD is presented. This metamodel is expressed in Alloy specification language [5], what allows for the analysis of the proposed metamodel using Alloy Analyzer tool.

Alloy is a declarative language for expressing structural constraints based on the first-order logic. An Alloy model is composed of sets (called 'signatures'), relations in these sets (represented as signatures' fields) and constraints over the sets and relations (called 'facts'). For detailed description, please refer to [5] or to the Alloy project Web site http://alloy.mit.edu.

The partial MAD metamodel is presented in Fig. 4 and Fig. 5. The main elements of the MAD models (represented as the `MAD` signature) are: decision problems (the `DecisionProblem` signature), requirements (the `Requirement` signature) and solutions (the `Solution` signature). Every single problem and solution in the MAD model must be in one of the possible states represented here as the singleton subsets of the `ProblemState` and `SolutionState` signatures (compare with Fig. 1).

```
sig MAD {}
sig DecisionProblem {}
sig Requirement {}
sig Solution {}

abstract sig ProblemState {}
one sig DefinedProblem extends ProblemState {}
one sig RequiresReassessment extends ProblemState {}
one sig Solved extends ProblemState {}
one sig BeingSolved extends ProblemState {}

abstract sig SolutionState {}
one sig DefinedSolution extends SolutionState {}
one sig Infeasible extends SolutionState {}
one sig Feasible extends SolutionState {}
one sig Chosen extends SolutionState {}
```

Fig. 4 Signatures

Fig. 5 shows the different possible relations between models, problems, solutions, requirements and problems' and solutions' states, that take place in MAD models. For example, the `requirements` relation defines the set of requirements relevant to the problem (graphically connected to the problem). The following additional constraints have been added as the Alloy facts:

1. If the problem belongs to the model all of its preceding problems and successors also belong to that model;
2. The "leads to" relation cannot form a cycle;
3. Not more than one of considered solutions for the problem can be in the "chosen" state;
4. If the problem has a preceding problem, the preceding one must have been solved in the past (and then its solution generated the next problem) and can never be again in the "defined" state;
5. The solved problem must have a chosen solution.

```
sig MAD {
    problems: set DecisionProblem,
    problemState: problems -> one ProblemState,
    leadsTo: problems -> problems,
    requirements: problems -> Requirement,
    solutions: problems -> (Solution -> one SolutionState)
}{
    all p: problems | p.(leadsTo + ~leadsTo) in problems    // 1.
    all p: problems | p not in p.^leadsTo                   // 2.
    all p: problems | lone s : Solution |
                           s -> Chosen in solutions[p]      // 3.
    all p: problems | all q: p.~leadsTo |
                      problemState[q] != DefinedProblem     // 4.
    all p: problems | problemState[p] = Solved =>
          one s: Solution | s -> Chosen in solutions[p]     // 5.
}
```

Fig. 5 The `MAD` signature details

For the above MAD metamodel, the decision consistency relation, which was defined in Sect. 5, can be written in the form of an Alloy predicate as in Fig. 6.

```
fun context[m: MAD, p: DecisionProblem]: Requirement + Solution {
    m.requirements[p] + {s: Solution | some q: m.problems |
      p in q.^(m.leadsTo) and s -> Chosen in m.solutions[q] }
}

fun solution[m: MAD, p: DecisionProblem]: Solution {
    {s: Solution | s -> Chosen in m.solutions[p]}
}

pred consistent [m1: MAD, m2: MAD] {
    all p: m1.problems & m2.problems |
         (m1.problemState[p] = Solved and
          m2.problemState[p] = Solved) =>
             (solution[m1,p] = solution[m2,p] or
              context[m1,p] != context[m2,p])
}
```

Fig. 6 Decision consistency relation

As it has been mentioned, Alloy specifications can be analyzed using the Alloy Analyzer tool. Such an analysis may be of two forms: *simulation*, which involves finding instances that satisfy a given property, or *checking*, which involves finding a counterexample—an instance that violates a given property. For the proposed metamodel, a number of instances have been generated with the Alloy Analyzer and these instances have been studied to assure both the correctness and completeness of the metamodel, i.e. that all the constraints have been captured and appropriately expressed.

7 Conclusion and Further Work

MAD notation has been originally developed as a simple tool for system architects to document architectural decisions [13]. It has appeared that this notation offers some constructions particularly useful for building the models in an iterative way, where new requirements appear after the whole or parts of the model were created [11]. In other words, the MAD notation may support the process of making changes in the software architecture during further evolution and maintenance of the software architecture.

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 [13]. The tool has been designed as a diagram editor being an extension to MS Word. Unfortunately, the main problem we faced trying to extend this tool to support the process of rebuilding models was the lack of a precise MAD metamodel. In this work, such a metamodel has been proposed. After creating a tool based on this metamodel, further empirical evaluation of the concepts presented here will be conducted.

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

References

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

[2] Bosch J, Jansen A (2005) Software Architecture as a Set of Architectural Design Decisions. Proc. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05): 109-120, IEEE Computer Society

[3] Capilla R, Nava F, Dueñas JC (2007) 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 : 9-16, IEEE CS Press

[4] Harrison NB, Avgeriou P, Zdun U (2007) Using Patterns to Capture Architectural Decisions. IEEE Software 24(4):38-45

[5] Jackson D (2012) Software Abstractions: Logic, Language, and Analysis, 2nd edn. MIT Press

[6] Jansen A, Avgeriou P, van der Ven JS (2009) Enriching Software Architecture Documentation. Journal of Systems and Software 82(8):1232-1248

[7] Kruchten P (2003) The Rational Unified Process - An Introduction, 3rd edn. Addison-Wesley

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

[9] Kruchten P, Capilla R, Dueñas JC (2009) The Role of a Decision View in Software Architecture Practice. IEEE Software 26(2):36-42

[10] Shahin M, Liang P, Khayyambashi MR (2010) Improving understandability of architecture design through visualization of architectural design decision. Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge:88-95, ACM

[11] Szlenk M, Zalewski A, Kijas S (2012) Modelling architectural decisions under changing requirements. Proc. Joint 10th Working Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA 2012): 211-214. IEEE CS

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

[13] Zalewski A, Kijas S, Sokołowska D (2011) Capturing Architecture Evolution with Maps of Architectural Decisions 2.0. In Crnkovic I, Gruhn V, Book M (eds) Proc. 5th European Conference on Software Architecture (ECSA 2011). LNCS 6903:83-96, Springer-Verlag

[14] Zimmermann O, Koehler J, Leymann F, Polley R, Schuster N (2009) Managing architectural decision models with dependency relations, integrity constraints, and production rules. Journal of Systems and Software 82(8):1249-1267