Integrating decision management with UML modeling concepts and tools

Könemann, Patrick

Published in:
Joint Working IEEE/IFIP Conference on Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009.

Link to article, DOI:
10.1109/WICSA.2009.5290824

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

Integrating Decision Management with UML Modeling Concepts and Tools

Patrick Könemann
Informatics and Mathematical Modelling
Technical University of Denmark
2800 Kgs. Lyngby, Denmark
pk@imm.dtu.dk

Abstract

Numerous design decisions including architectural decisions are made while developing a software system, which influence the architecture of the system as well as subsequent decisions. Several tools already exist for managing design decisions, i.e. capturing, documenting, and maintaining them, but also for guiding the user by proposing subsequent decisions. In model-based software development, many decisions directly affect the structural and behavioral models used to describe and develop a software system and its architecture. However, the decisions are typically not connected to these models.

In this paper, we propose an integration of a decision management and a UML-based modeling tool, based on use cases we distill from an example: the UML modeling tool shall show all decisions related to a model and allow extending or updating them; the decision management tool shall trigger the modeling tool to enforce design decisions (modify the models). We define tool-independent concepts and architecture building blocks supporting these requirements and present first ideas how this can be implemented in the IBM Rational Software Modeler and Architectural Decision Knowledge Wiki. This seamless integration of formerly disconnected tools could improve tool usability as well as decision maker productivity.

1. Introduction

Complex software systems are often built using several models on different levels of abstraction, as described by the model driven architecture [1]. UML is a language for those models, and tools like the IBM Rational Software Modeler[1] (RSM) or Borland Together assist the user in authoring them and also provide code generation facilities. Current decision management tools like the Architectural Decision Knowledge Wiki [2] and the Architectural Design Decision Support System (ADDSS) [3] help documenting, understanding, and developing software systems. Their primary goal is to capture, document, and maintain design decisions and also support users in making decisions during the development process. However, they do not yet connect decisions to the actual design models. The contributions of this paper are concepts for connecting these artifacts.

This paper first presents an example which shows how design decisions can be exploited for modeling (the full example and all concepts in more detail are given in [4]). Then use cases for the interaction between a decision management and a UML modeling tool are presented to motivate their integration. The last section presents a design for this integration such that enforced decisions can be propagated and applied to the models, and that the modeling tool makes use of design decisions. That is, it shows information about enforced design decisions, provides functions for enforcing decisions, and for creating new decisions.

We refer to the RSM as the modeling and to the Architectural Decision Knowledge Wiki as the decision management tool; though our concepts are also realizable with other tools.

1.1. Terms

This section defines important terms. A *design model* is a formal model used for the development of a software system; one modeling language for this purpose is the UML. A *design decision* describes changes in one or more design models due to a particular problem; it includes structural model changes, rationales, and consequences for further design decisions. We distinguish between the *issue*, *alternatives*, and *outcomes* of design decisions, as introduced in [5]. An *issue* is a description of a particular design problem which is expressed in a formal decision model in the decision management tool; it contains alternative solutions, called *alternatives*. An *alternative* is a particular solution for a design issue; it contains information about its advantages and disadvantages and optionally a *decision structure*. Issues and alternatives may have a *scope*, which describes the relevant design model elements. A *decision structure* describes changes in a design model (similar to a design pattern). An *outcome* is a made decision for an issue; it contains a justification and points to the selected alternative; it might also refer to participating design model elements according to the *decision structure*.

Both issues and alternatives, are independent from any project – they describe the problem and possible solutions

1. http://www.ibm.com/software/awdtools/modeler/swmodeler/

978-1-4244-4985-9/09/$25.00 ©2009 IEEE

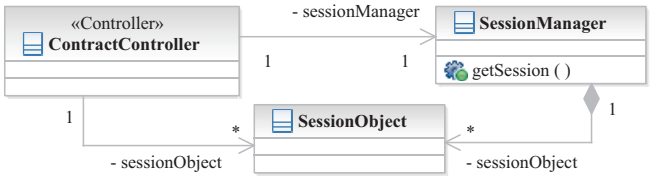

Figure 1. Design model elements for decision 4

in general. An outcome, on the other hand, is the result of a decision made for some issue in a particular project.

2. Example

A case study in [4] describes a model-based development of a simple web application which allows users to create and store customer contracts. We use one example here to point out the use and re-use of design decisions. The principles of design decisions are applicable to any application genre, although the example refers to a web application.

Preceding decisions for the following example already decided to use a layered architecture and the MVC (model-view-controller) pattern in the presentation layer of the application, which resulted in UML component- and class-diagrams. The following decision 4 deals with session management (a brief issue description is in italics followed by three alternatives):

Decision 4: Session Management
Specify the strategy for managing the session as proposed by Fowler in "Patterns of Enterprise Application Architecture".
Server Session State: The server is responsible for the session state and thus requires a session object to store it; in use with the MVC pattern, the controller is responsible for the session object.
Client Session State: The client is responsible for handling the session data and thus it is transferred in each request; this makes the server completely stateless.
Database Session State: The state is stored in a database and thus queried and processed in each server request.

The choice of the user is the *Server Session State*. Figure 1 shows the relevant design model elements: *SessionObject* and some references were added because of decision 4; due to prior decisions and work on the models, *ContractController* and *SessionManager* already exist in the model; so we do not want to create new components but rather re-use the existing ones.

3. Requirements

Based on the example, we distill the main use cases for connecting design decisions and design models in the development process. They are categorized for the modeling and the decision management tool.

3.1. The Modeling Tool

The use cases are generic and apply to any modeling tool, not only those based on UML. References to the example are stated in *italics*.

Use Case M1: Show design decisions
All project-related design issues are browsable from within the modeling tool although they are technically located in the decision management tool. Issues, alternatives (not necessarily related to already made decisions), and outcomes are shown.
Decisions 1 to 3 from [4] are shown as made decisions; "Session Management" is shown as a currently relevant issue.

Use Case M2: Enforce an outcome for an issue
An alternative is selected for the enforcement of an outcome. If the alternative has a decision structure, the model has to be changed accordingly. The action can be triggered from use cases M1 and D2, and consists of two steps: first, the participating design model elements are selected; second, the design model is changed. Afterwards, a binding (stored in the library) links the respective design model elements to the outcome.
The existing ContractController *and* SessionManager *are selected and* SessionObject *is newly created. All three components are then bound to the outcome.*

Use Case M3: Show design model elements for design decisions
The selection of design decisions in use case M1 or D1 results in a context-sensitive list of design model elements: a) show all design model elements bound to a particular outcome, i.e. all design model elements which are involved in a decision; b) show all design model elements which are relevant (in the scope) of a selected issue or alternative; c) show the decision structure of a particular alternative, independent of the current design model; this shows the pattern to the user without already making the decision.
A selection of decision 4 highlights the design model elements shown in Fig. 1.

Use Case M4: Show design decisions for design model elements
A selection of design model elements shows all design decisions in which they occur. Moreover, all open issues in the project can be queried for which the selected design model elements are contained in the scope; since this might be a time-consuming operation, it must be triggered explicitly.
A selection of SessionObject *highlights decision 4.*

Use Case M5: Link design model changes to design decisions
The user can record changes in the design model for documentation and re-use. To this end, the decision structure is computed and the user may attach it either to an existing alternative of an existing design issue, or to a new alternative (which has to be created) of an existing design issue, or to

a new alternative and a new design issue (both have to be created). In the end, the respective design model elements are bound to the newly created outcome (cf. use case M2). *An example is given in [4].*

Use Case M6: Check consistency
A consistency check validates the bindings between design model elements and decision outcomes. In case of invalidity, the user is informed. This function shall be used by use cases M1 to M5.
For decision 4, the existence of its outcome and of all elements in Fig. 1 is checked.

3.2. The Decision Management Tool

Use Case D1: Show design model elements
For issues, alternatives, or outcomes, show the respective design model elements. This is done by triggering use case M3 in the modeling tool.

Use Case D2: Enforce an outcome for an issue
If a decision is made, i.e. an alternative of an issue is chosen for an outcome, the modeling tool is requested to change the design model accordingly. Hence, use case M2 is triggered.

4. Integrated Design

This section introduces a design and in particular component responsibilities for integrating the two kinds of tools, based on the use cases from Sect. 3. We assume that the tools are extendable concerning data access and user interaction. We made some important design decisions for this design: UML is used as the modeling language since it is commonly used in the domain of software engineering; the decision management tool and the decision model as well as the modeling tool and the design model are (and remain) independent from other components; all additional user interactions will be integrated into the existing tools, i.e. no new tools are introduced.

Figure 2 shows both tools along with two new components: the *DecisionStructureLibrary* is responsible for storing decision structures as well as their bindings to the design and decision models; the *DecisionStructureManager*, responsible for coordinating all communication, can be seen as a facade for the modeling tool and the library. In other words, the library contains all decision structures as well as the connection between decisions and design model elements, and the manager connects all components. The interfaces are explained in more detail next.

4.1. Modeling Tool Interfaces

Besides the usual functionality for authoring UML models, we require the following two interfaces for the modeling tool: *IModelAccess* must provide full access to the models and to context-sensitive information (for use cases M2 to M6), e.g. currently selected elements; *IToolGui* must provide the opportunity to extend the GUI (for use cases M1 to M6), in particular, user interactions must be integrated.

4.2. Decision Management Tool Interface

The role of a decision management tool is to create and maintain decisions and their relations, their consequences, and further informal information. To reflect the status of a decision, an outcome must either be *undecided*, *decided*, or *enforced*; the design model is about to be changed with respect to a decision if its status is *decided*. Both use cases D1 and D2, only delegate the functionality to the modeling tool; hence, their integration into the decision management tool is optional.

However, concerning access to the decision model, we require the following functionality: *get list of issues and alternatives* independently of a project, and also *outcomes* for a particular project, or by their id, or all *related issues* of a given issue (for use cases M1, M3, M4, and M6). *Create a new issue* by providing all required attributes; *create a new alternative* for a given issue by providing all required attributes; *create a new outcome* for a given issue and alternative, providing also a justification (for use case M5). *Set status of an outcome* (for use case M2).

4.3. Decision Structure Manager Interface

The decision structure manager coordinates the interaction between both tools and has the following responsibilities at the interface for the decision management tool: *show the scope of an issue*, i.e. all model elements that might be affected by that issue; *show the decision structures* for a particular alternative; *show the related design model elements* of a particular outcome (for use case D1). *Perform a decision enforcement* and change the design model according to the given decision structure (for use case D2).

These functions provide sufficient information for the decision structure manager to fulfill the use cases. The decision structure library contains the decision structures for alternatives as well as bindings from outcomes to the respective design model elements. So the interface provides an API for: *storing and retrieving decision structures* of alternatives; *storing, retrieving, and updating bindings* between outcomes and design model elements.

5. Related work

There are several systems and approaches besides the Architectural Decision Knowledge Wiki which support users in capturing and making decisions during a software development process. ADDSS [3], for instance, is a web-based tool to collect, store, and provide architectural knowledge, but no automated re-use involving the participated design

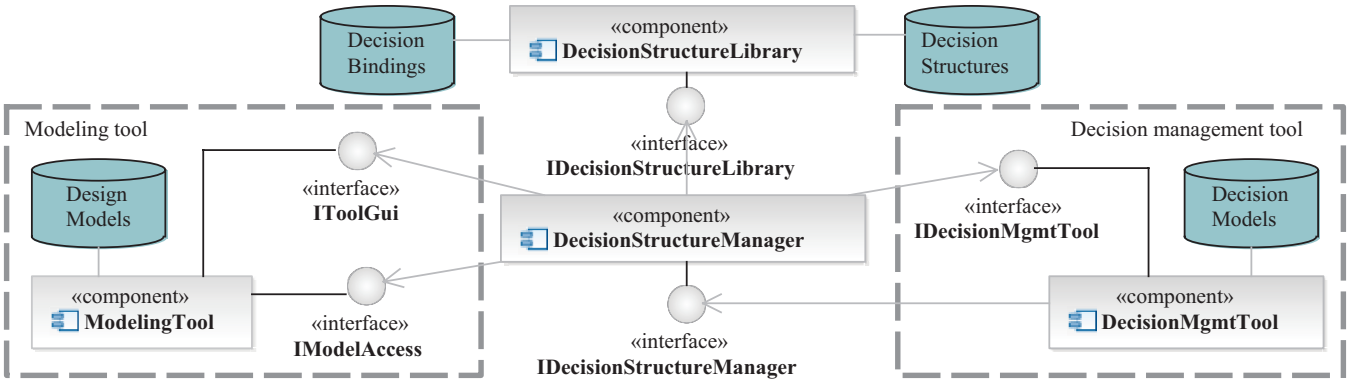

Figure 2. The overall design of the integration of a decision management and a modeling tool

models is supported. AREL [6] is another system for the documentation of architectural decisions based on their rationales. It introduces a UML profile for modeling architecture rationales and traces them back to the architectural elements. But its intention is not to capture and re-use changes in the architectural artifacts. The Archium language [7] combines the implementation with decision making, but not design models. To conclude, there are many tools (also [8], [9]) for documenting and reasoning about decisions and capturing knowledge, but the integration with models of a model-based development process is not covered by any system yet.

Some modeling tools like RSM and Borland Together provide pattern authoring capabilities similar to the intention of decision structures. However, a meta model for expressing relations between them as well as tool supported guidance, i.e. proposing subsequent patterns, is missing.

6. Summary

The example motivated the connection of design decisions (and thus existing design knowledge) and the design models. The development process is extended by proposing decisions (use case M4), supporting the user in making decisions (use case M2), and making the relation between design decisions and design models explicit (use cases M1 and M3). The user has therefore immediate access to all design decisions and design knowledge directly in the modeling tool. Furthermore, often recurring decisions and patterns in the models can be stored and re-used (use case M5), which can be seen as best practises. Hence, their application is easier and faster, and less error-prone. An initial prototype already implements the interfaces; decision representation in the modeling tool, decision structure handling, and user-interaction are future work.

Acknowledgment

The ideas in this paper were compiled with Olaf Zimmermann at a workshop at the IBM Research Lab in Zurich in December 2008. Many thanks to Ekkart Kindler for lots of helpful discussions and advices.

References

[1] Object Management Group: MDA Guide v1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01 (2003).

[2] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster. Reusable Architectural Decision Models for Enterprise Application Development. In S. Overhage, Clemens A. Szyperski, R. Reussner, and J. A. Stafford, editors, *QoSA*, LNCS 4880, pp. 15–32. Springer, 2007.

[3] R. Capilla, F. Nava, and Juan C. Duenas. Modeling and Documenting the Evolution of Architectural Design Decisions. In *Proceedings of the 2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, p. 9, IEEE CS Press, 2007.

[4] P. Könemann. Integrating a Design Decision Management System with a UML Modeling Tool. IMM-Technical Report-2009-07, Technical University of Denmark, April 2009.

[5] O. Zimmermann. An Architectural Decision Modeling Framework for Service-Oriented Architecture Design. PhD Dissertation, Universitt Stuttgart, Germany. dissertaion.de, 2009.

[6] A. Tang, Y. Jin, and J. Han. A Rationale-based Architecture Model for Design Traceability and Reasoning. *Journal of Systems and Software*, 80(6): pp. 918–934, 2007.

[7] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer. Tool Support for Architectural Decisions. In *WICSA*, p. 4. IEEE Computer Society, 2007.

[8] F. Bachmann and P. Merson. Experience Using the Web-Based Tool Wiki for Architecture Documentation. Technical Report CMU/SEI-2005-TN-041, Carnegie Mellon University, 2005.

[9] P. Liang, A. Jansen, and P. Avgeriou. Knowledge Architect: A Tool Suite for Managing Software Architecture Knowledge. Technical Report RUG-SEARCH-09-L01, University of Groningen, 2009.