Proceedings of the IASTED International Conference
SOFTWARE ENGINEERING
February 17-19, 2004, Innsbruck, Austria

A CONCEPTUAL MODEL FOR REQUIREMENTS ENGINEERING AND MANAGEMENT FOR CHANGE-INTENSIVE SOFTWARE

Jewgenij Botaschanjan, Andreas Fleischmann, Markus Pister[1]
Technische Universität München, Institut für Informatik
Lehrstuhl für Software and Systems Engineering
Boltzmannstr. 3, 85748 München, Germany
{botascha, fleischa, pister}@in.tum.de

Abstract

In the development of software with evolving requirements, activities of requirements-engineering and management are present through the whole software development process and affect most of the actors involved. This paper presents a conceptual model which aims to an efficient requirements management by offering a central data structure for requirements that integrates requirements, design and implementation workflow. This conceptual model was developed for change-intensive embedded systems, but can be easily adapted to other domains. This paper presents the model itself, techniques to adapt this model to specific needs, and demonstrates its usage.

Key Words

Software Requirements, Software Evolution, Requirements Tracing, Software Product Lines

1 Introduction

Requirements engineering is a key activity in the development of software systems. One of the challenges requirements engineering has to pass is the management of a specification document. This task becomes especially critical when developing change-intensive software systems and product lines. The specification document has to be readable, yet concise (despite containing a large amount of requirements), has to offer the appropriate views on the information to all the different stakeholders (including designers, testers, project managers, customers), and has to be constantly synchronized with the other development products (such as design, code, release plans). The conceptual model presented in this paper especially supports those tasks by offering a central data structure, which serves as an interface between the involved participants such as requirements engineers, designers, release planners and implementers.

This conceptual model was developed for evolutional embedded real-time systems in the context of the European EMPRESS project [1]. It consists of a central data structure and methods for its usage. This paper focuses on the description of the data structure; the application of the model is demonstrated along the presented examples.

The core of the central data structure is the part concerning the specification, providing the stakeholders with structured information about the system to be developed. Other elements of the data structure capture information about the design, release planning and implementation and relate this information to the specification. This integration of requirements, design and implementation into one model allows us formulating comprehensive queries and constraints that especially support tasks involving several workflows such as consistency-checks, change-management and tracing. This integration serves as a basis for an integrated tool support.

The next two sections present the data structure as it has been developed for change intensive embedded systems (section 2), and show how to adapt this model to specific domains (section 3). The last section gives a brief summary of the results and sketches further research plans.

2 A Central Data Structure for Change-Intensive Embedded Systems

The data structure for change-intensive requirements engineering is segmented in three parts: a specification part, a design interface and an implementation interface. The model consists of elements and relationships between them which are depicted as UML class diagrams [2].

[1] This work was sponsored by the EUREKA-ITEA project “EMPRESS” (ITEA 01003)

2.1 Specification Part

A specification is to document requirements. Since specifications can get rather large (e.g. several thousands of requirements) and are used by various stakeholders (e.g. customers, testers, designers), a conceptual model for requirements has to offer flexible structuring mechanisms, providing the involved stakeholders with the right amount and abstraction level of information. The specification also has to be able to capture a whole product family and offer methods to extract single products from this family. These tasks are supported by the specification part. Its elements are shown in Figure 1.

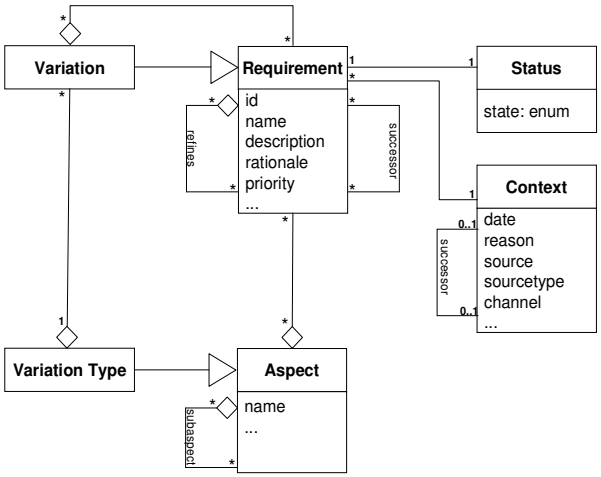


Figure 1: Specification Part of the Model

The following subsections explain these elements and their functions in detail.

2.1.1 Capturing Requirements

In the centre of the specification part stands the element "Requirement", storing all relevant information about a single requirement. The details about what contents and attributes of an individual requirement have to be stored (e.g. identifier, name, description, rationale, priority and so on) are not determined, so the user can use established templates such as [3].

A requirement's "Status" denotes the state of a requirement, that is, whether it is proposed, rejected, approved or deleted. The state space can be adapted, in that new states can be added such as "waiting room" or "in work" [4].

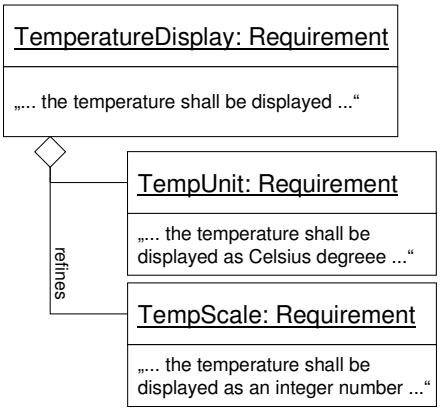
Each instance of a requirement is linked to a "Context" which describes the circumstances of its creation, change or deletion. A context contains for example information about the person ("source") who suggested a change, his role ("sourcetype"), such as "customer" or "management", the date, the reasons, and the channel (e.g. "interview", "contract", "phone call").

Contexts are a mechanism to realize Pre-Requirements Specification Traceability as demanded by [5]. With contexts it is possible to query the specification like "Show all requirements that origin from Mr. Smith", "Show all requirements that were added by the workshop at December 2003" or "Show the changes of the last three weeks". Since contexts are linked to each other by a "successor" association, they form a qualitative timeline for the whole specification. In combination with an elementary configuration management (which is not covered in this paper, for details see [6]), contexts allow to comprehend the evolution of requirements and revoking changes.

2.1.2 Structuring Requirements

The conceptual model provides two mechanisms to structure the specification: firstly, a refinement hierarchy, and secondly, aspect hierarchies.

The "refinement" association between requirements structures the specification by connecting requirements that state the same functionality on different levels of detail [7]. For example, a high-level requirement "the temperature shall be displayed" can be refined to more precise requirements that specify whether the temperature shall be displayed in Fahrenheit or Celsius and so on (see Example 1). The resulting structure of the refinement is an acyclic directed graph, whose nodes are requirements.



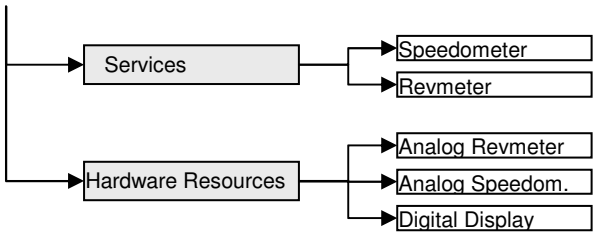
Example 1: Refinement

The refinement hierarchy offers a special view on the specification document. However a specification serves for several purposes and thus different structuring hierarchies are required. The concept of "Aspect" allows establishing these hierarchies along several viewpoints. Such viewpoints might be functional units, physical units, or non-functional requirements. Thus, an aspect is a set of requirements. A requirement can belong to several aspects, and aspects can form hierarchies by the "sub-aspect" association between them.

In order to improve the manageability of the document, correlations between requirements have to be captured

explicitly. These can be realized as particular associations between requirements [8]. However the effort to maintain such associations is high. Aspect hierarchies are an alternative concept for modeling of these correlations. An aspect aggregates all requirements involved in a particular relationship; this is not as precise as individual relations between single requirements, but it is easier to overview and manage.

The aspect hierarchies are domain specific and are customized for a particular application. For the domain of embedded real-time systems, an aspect hierarchy framework has been proposed in [6]. Example 2 shows some exemplary aspect hierarchies of it.



Example 2: Part of the Aspect Hierarchy for an Instument Cluster

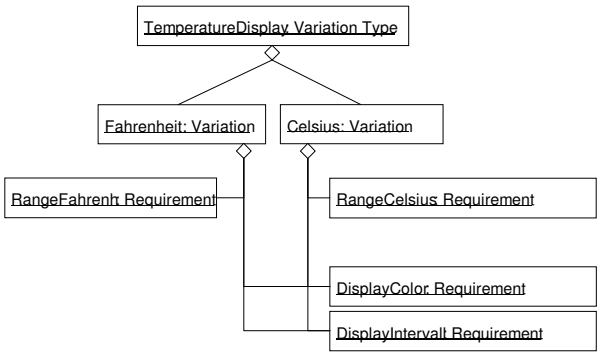
Aspects are a powerful concept to structure and query a specification. Aspect hierarchies allow queries that merge or intersect different aspects. Hence, queries can be performed such as "Show all requirements that define the allocation of the hardware resource “Digital Display” by the “Revmeter” service ".

2.1.3 Product Line Support

The conceptual model is able to capture product lines. Therefore, the elements "Variation" and "Variation Type" are introduced (left part of Figure 1).

Variations are special requirements (indicated by the inheritance between "Variation" and "Requirement") that are allowed to contradict each other in regard to a variation type. Example 3 demonstrates their usage.

Since a variation type inherits from aspects, it aggregates requirements; in Example 3, the variation type "TermperatureDisplay" aggregates all requirements regarding the temperature display. Also, a variation type contains two or more variations ("Fahrenheit" and "Celsius" in Example 3). A variation aggregates those requirements that have to be considered if the variation is chosen. Hence, the sets of requirements of different variations need not to be distinct.



Example 3: Variations and Variation Types

2.2 Design Interface

This interface serves traceability and tracking purposes for the design. It makes it possible to check whether all requirements have been considered in the design, and supports estimating the impact of a change in the specification on the design. Figure 2 shows the elements of the conceptual model that are involved in this interface.

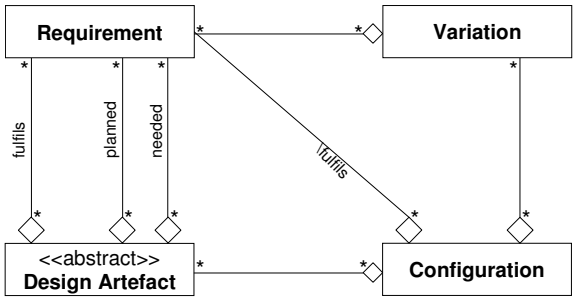


Figure 2: Design Interface of the Conceptual Model

The conceptual model offers an abstract element "Design Artefact" that must be adapted to the used design paradigm, for example to components and interfaces (see Figure 3, and Example 7).

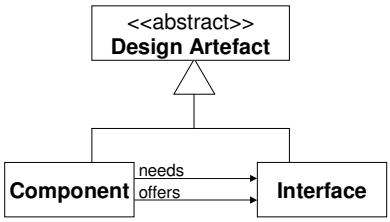
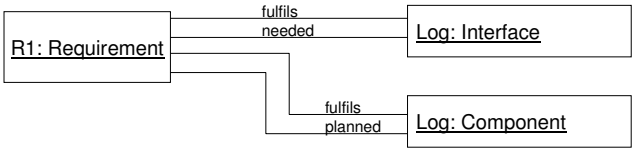


Figure 3: A possible Adaptation of "Design Artefact"

Then the designer distributes the requirements to the design artefacts, using the following associations: "fulfils" denotes which requirements are generally fulfilled by a design artefact. Those requirements can be either "planned" in the design artefact itself or delegated to other design artefacts, thus be "needed".

Example 4 shows an example for those associations, using the adaptation of "Design Artefact" as introduced in Figure 3.



Example 4: "fulfils", "needed" and "planned"

This connection between requirements and design supports the formulation of constraints to assure, for example, that all requirements are considered in the design (see the example in 2.4). It also allows a tracing from requirements to design elements, and vice versa.

Since the design elements can be adapted to the used design model, it is possible to connect this design interface with a design modelling tool such as Rational Rose or AutoFocus (for actual work on this, see section 4).

With the help of variations and variation types, a whole product family can be described within one instance of the conceptual model. In order to describe one particular product of this family, for all variation types at most one variation can be chosen. Such a decision is captured in the element "Configuration". Thus, a configuration represents one product of a product family. It consists of a set of variations and a collection of design elements to be implemented. The "fulfils" set of a configuration consist of the requirements of the "fulfils" sets of the aggregated design artefacts including the chosen variations.

Since the design interface establishes a connection between the specification and the design, it supports the estimation of the impact of a change in the requirements by pinpointing the affected design artefacts. This support is enhanced by the tracing to the implementation, as described in the following section.

2.3 Implementation Interface

The implementation interface allows tracing the requirements to the implementation, monitoring their implementation state, checking the consistency between the design and its implementation and supports the estimation of the impact of a change in the requirements. It builds on the element "Design Artefact" and adds realization states and release planning to the model. Figure 4 shows the elements involved in this interface.

For the release planning of an iterative implementation of the specified product, the conceptual model provides an element "Release". Each release relates to a single design artefact (for example, a class, an interface, or a component) of which it implements a subset, denoted with an aggregation "implements".

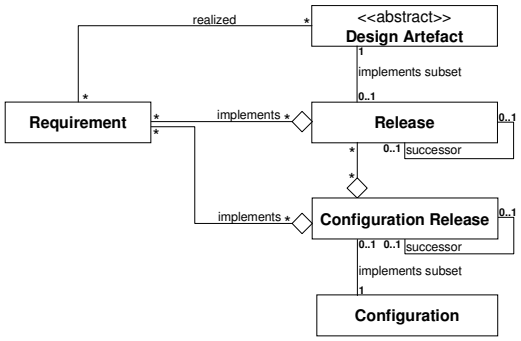
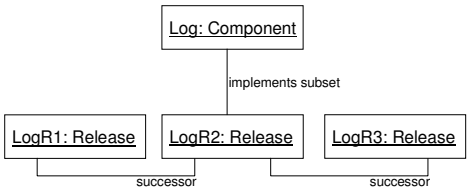


Figure 4: Implementation Interface of the Conceptual Model

A release plan is a sequence of releases, connected by the "successor" relationship. The current release, which is in progress, is indicated by the link "implements subset" to the design artefact. Example 5 shows an example for a release plan of a component, in which LogR2 is the current release.



Example 5: A Release Plan

In Figure 4, there is also a new relationship between requirements and design artefacts introduced: "realized". In contrast to the relationships “implements”, which serve for planning, the “realized” association is used to track the implementation status. As soon as a requirement has been implemented for a design artefact, it's moved from its "planned" set (see section 2.2) to its "realized" set. Note that the realization state of a requirement is not captured in its "Status" because a requirement might be involved in several design artefacts, thus having several realization states.

While a "Release" represents one release of a single design artefact, a "Configuration Release" represents a release of one product of the product family. The associations of the configuration release are analogous to that of the release.

The connection from requirements to the implementation enhances the support of the estimation of change impact described in the previous section: when a set of requirements is to be changed, the links pinpoint the affected design elements, and the release plan states whether actual implementation or only release plans have to be changed.

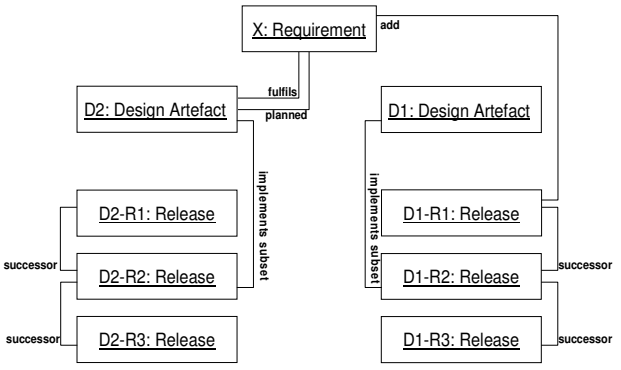
Since each design artefact declares which requirements it fulfils (by the "fulfils" association) and since each release defines a set of requirements it implements (by the "implements" association), those two sets can be used to reveal inconsistencies of the release planning as well as of the design. For example, queries can be answered such as

"Are there requirements that have not been assigned to a release or to a design artefact?", "Does a design artefact declare to fulfil requirements, which its releases don't declare to implement?", or "Does a release declare to implement requirements, which don't belong to its design artefact?" As another example, the data structure can also help to reveal inconsistencies when a design artefact delegates parts of its requirements to other design artefacts, and somewhere in this delegation process a requirement gets lost. The next section, 2.4, shows an example of such an interaction between the three parts of the conceptual model.

2.4 Interaction of the Three Parts

This subsection presents a small example to demonstrate the interaction of the three parts of the conceptual model and to show how it supports the cooperation of the involved stakeholders: both by monitoring consistency constraints of the model, and by structuring the communication paths between requirements engineers, designers, release planners, and implementers.

Consider a change in the design (e.g. moving a requirement from one design artefact to another). When the designer moves a requirement from one design artefact to another (that is, by moving the associations "fulfils", "needed", and "planned"), the requirement is still linked to a release of its old component. This situation results in temporary inconsistencies, which are shown in Example 6.



Example 6: Temporary Inconsistencies in the Middle of a Design Change

Example 6 shows the situation after a requirement X has been moved from one design artefact (D1) to another (D2). The current releases are D2-R2 and D1-R2. Since X has been added to D1-R1, it has already been implemented in the design artefact D1. Since it has been newly moved to D2, it hasn't been implemented yet; thus, it's in the "planned" set of that design artefact. And because the release planning hasn't been updated yet, the model signals two temporary inconsistencies:

- "According to the release plan, X is implemented for D1, but D1 doesn't claim to fulfil X."
- "D2 claims to plan X, but X isn't added to any of its releases."

Consequently, the requirement has to be explicitly removed from the old release (thus signalising the affected implementers that parts of the code might get abandoned if the requirement has already be implemented) and added to the other component's release-plan (thus signalising the affected implementers that new code might have to be written).

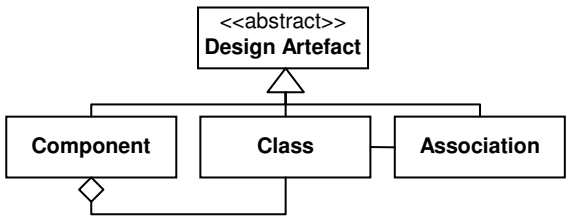
3 Adaptation of the Model to Specific Needs

The data structure presented in the previous section comes along with a set of refinement methods for adapting the model to specific needs, for example, to a specific domain, to a certain design paradigm or to a particular tool.

One refinement method is "specialization", which means that elements of the structure can be extended (likewise the mechanism of inheritance) to map a more specific view on the application area to the data structure. The other method is "tailoring", which is used to cut off elements or relations from the data structure. The following subsections present two examples of how the data structure can be adapted to specific needs.

3.1 Adaptation to a Design Paradigm

In order to ease and eventually automate the synchronisation of the design and its representation within the conceptual model, it is required to adapt the design interface to the applied design paradigm. Therefore the design interface consists of an abstract element "Design Artefact" which must be refined to match the used design paradigm. With this, a seamless integration of the conceptual requirements model and a design modelling tool is possible. If the design is modelled using UML, an excerpt of the adaptation could look like Example 7. The granularity of the representation of the design paradigm in the conceptual model should be chosen in respect to the manageability of links.



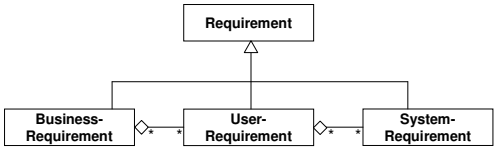
Example 7: An Adaptation to the UML Design Paradigm

Beside the synchronisation between the conceptual model and a design tool, this adaptation makes it also possible to formulate and check consistency constraints between design artefacts. In Example 7, a constraint between

classes and interfaces could be "A class fulfils all requirements expressed by the specification its interface."

3.2 Adapting the Requirements Structure

In section 2.1.2, two general structuring mechanisms had been introduced: refinement and aspect hierarchies. There exists a variety of more specific approaches [4, 10]. For example, a specification can be written top-down, deriving user requirements and system requirements from high-level business requirements [4]. The conceptual model can be adapted to this top-down practise. The different levels of details can be modelled in the data structure using the "refinement" association described in section 2.1.2. Furthermore, the model can be adapted by introducing new types of requirements as shown in Example 8.



Example 8: Introducing Levels of Details for Requirements

With this adaptation, more precise consistency constraints can be formulated such as "Are there user requirements, which have not been passed on to system requirements?" or "Are there system requirements that don't relate to a business goal?"

4 Conclusion and Future Work

The developed conceptual model uses a common data structure to integrate the requirements workflow with the design and implementation workflows. The data structure can be adapted to fit with specific tools, paradigms, and domains; for example it can be combined with the Volere Requirements Template [3], and AutoFocus design tool [11]. This integration enhances the dataflow between the workflows, allows pre- and post-tracing within a specification and from requirements to design/code and permits the formulation and automation of consistency checks. The notion of Aspects is used as an easy manageable mechanism to model relationships between requirements and as a powerful concept to structure and query specifications. Aspect and refinement hierarchies need some effort to apply, which can be decreased by providing aspect hierarchy frameworks (for example [6] for the domain of embedded systems), and which pay off with change-intensive systems.

Handling a noteworthy amount of requirements needs tool support. Therefore, a prototypical tool called ESTA is under development, using Java and mySQL. It should support the use of the conceptual model and connect it with the design modelling tool AutoFOCUS [11, 12].

Right now, the usage and the efficiency of the conceptual model is tested in the context of the EMPRESS project within a case-study of DaimlerChrysler. The results of this study, as well as the complete documentation of the conceptual model will presumably be published in the context of the EMPRESS project in the first term of 2004 as part of an overall development process for evolutional embedded real-time systems [1, 6].

References

[1] EMPRESS Homepage, *http://www.empress-itea.org*

[2] M. Fowler, *UML Distilled. A Brief Guide to the Standard Modelling Language* (München, Germany: Addison-Wesley, 1999)

[3] J. Robertson, S. Robertson, Volere Requirements Specification Template, Edition 9, 2003
www.volere.co.uk

[4] K. E. Wiegers, *Software requirements* (Redmond, WA: Microsoft Press, 2003)

[5] O. Gotel, A. Finkelstein, An Analysis of the Requirements Tracing Problem, *Proc. International Conference on Requirements Engineering 1994,* IEEE CS Press, 1994, 94-101

[6] J. Botaschanjan, A. Fleischmann, M. Pister, Integration of Classifications, Structuring and Process Models, *Framework to Requirements* (Chapter 5), EMPRESS Compositum, to be published in 2004

[7] A. Gabb, The Requirements Spectrum, *Proc. First Regional Symposium of the Systems Engineering Society of Australia 1998 (SE98)*, Canberra/Australia 1998, http://www1.tpgi.com.au/users/agabb/Files/Spectrum-SE98.zip

[8] P. Leitelier, A Framework for Requirements Traceability in UML-based Projects, *1st International Workshop on Traceability in Emerging Forms of Software Engineering, In conjunction with the 17th IEEE International Conference on Automated Software Engineering*, U.K., 2002

[9] ISO/IEC 9126-1:2001(E), Software Engineering – Product Quality – Part 1, 2001

[10] I. Summerville, *Software Engineering*, (Boston, MA: Addison-Wesley, 6th edition, 2000)

[11] F. Huber, B. Schätz, K. Spiess, AutoFocus - Ein Werkzeugkonzept zur Beschreibung verteilter Systeme, *Arbeitsberichte des Instituts für mathematische Maschinen und Datenverarbeitung*, Vol. 29, Nr. 9., University Nürnberg, 1996, pages 165-174

[12] F. Huber, J. Philipps, O. Slotosch, Model-based Development of Embedded Systems, *Proc. Embedded Intelligence,* Nürnberg, 2002