T U M

I N S T I T U T F Ü R I N F O R M A T I K

Towards an Integrated Approach to Requirement Engineering

Manfred Broy, Andreas Fleischman, Shareeful Islam, Leonid Kof, Klaus Lochman, Christian Leuxner, Birgit Penzenstadler, Daniel Mendez Fernandez, Wassiou Sitou, Sebastian Winter

TUM-I0935
Dezember 09

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-12-I0935-0/1.-FI
Alle Rechte vorbehalten
Nachdruck auch auszugsweise verboten

©2009

Druck: Institut für Informatik der
Technischen Universität München

Contents

1 Introduction

1.1 Motivation

Requirements Engineering (RE) still constitutes the Achilles' heel of the entire software development activities. Even worse, there is a considerable gap between the state of the art in academic RE and in industrial RE. Although many academic RE approaches exhibit a high potential in practical application, they are not applied in practice.

This report demonstrates that a set of approaches developed at the Technische Universität München can be integrated into one approach that covers most RE tasks. The aim of this integrated approach is to provide a comprehensive RE framework. In industrial practice, RE is influenced by a plethora of stakeholders. This makes it difficult to address the needs of *every* stakeholder and not loosing the big picture.

The presented artefact-oriented approach provides a guideline that defines what should be specified so that the resulting set of requirements is as complete as possible. The approach is supported by adequate modeling techniques required to achieve the objectives of consistency and completeness in requirements specifications. The approach includes both general technical tasks, as proper writing of requirements documents or transition from requirements to design, and methodical issues, as structuring of requirements artefacts.

1.2 An Overall Approach to Integrated Requirements Engineering

The overall approach is steered by the paradigm of *artefact-oriented requirements engineering* and is depicted in Fig. 1.1. The starting point that integrates the different methods and techniques is a meta model for artefacts orientation. The meta model defines terminology, structure, and semantics of the paradigm. Additionally, it specifies in an abstract manner how to define requirements engineering approaches that follow this paradigm. Artefact-orientation in general is based on the philosophy of structuring the produced requirements specifications according to a given common result model and thereby easing access and overview of the process.

The artefact models guide the elaboration of the result models in individual projects as they define possible concepts and syntax. The concepts describe those aspects of a system under consideration dealt with during the development process [Schätz et al., 2002]. How the models are defined depends on their concerns, e.g. concerning the coupling of different tools for different development aspects like aspects of use cases and their relation to aspects of service modelling [Endres and Rombach, 2003, Schätz et al., 2002]. The models usually formulate and structure domain-specific description techniques by characterising the concepts of this domain,

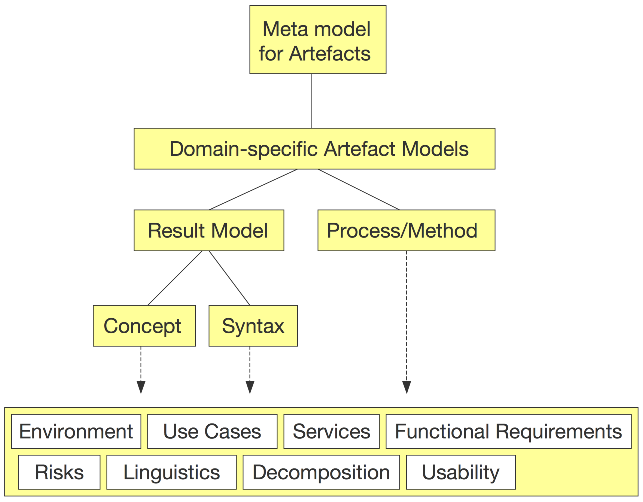


Fig. 1.1: The overall artefact-based requirements engineering approach.

reflected by the elements and their relations found in the description techniques [Schätz, 2008]. The syntax propositions inter alia give instructions on how to document the concepts. For instance, which notion to take for representing system's services or use cases.

Hence, according to this view, the artefact models are all situated in a specific domain of application. We propose in the context of this report two artefact models: one of them covering aspects of embedded systems and the other one aspects of business information systems.

Each artefact model furthermore requires a set of (domain-specific) methods, i.e. a process. A method defines how to use and produce an artefact as part of an overall process description. For example, how to define use cases and how to manage risks that arise from the use case descriptions.

Finally, we benefit from artefact-orientation by obtaining

1. the definition of a clear terminology for the produced contents of the results and used processes;

2. the definition of a (syntactical) consistent requirements specifications that can be customised to individual needs of single projects (i.e. by tailoring the artefact model); and

3. the definition of a tool box or construction kit with compatible building blocks that define for single artefacts the detailed concepts, notions and methods.

With respect to process integration, the framework is given by the V-Modell XT [Friedrich et al., 2009].

1.3 Case Studies

The feasibility of the different parts of the proposed integrated approach to requirements engineering is demonstrated by means of case studies. Thereby, two case studies are central:

The first one is about an adaptive cruise control (ACC); such an ACC controls the speed of a system (constant speed control), and, if a car ahead is discovered, controls the speed in dependency of the distance to the car ahead (follow-up control). The full case study can be found in [Feilkas et al., 2009a].

The second one is about a smart key entry system (SmartKeyEntry-System); such a SmartKeyEntry-System is characterized by the main goal of enhancing the convenience of comfort services of locking and unlocking the car. Further goals include the remote control functions using wireless key device, entry and start functions without having to use a wireless key device, and customization of remote control functions according to user's preferences. The full case study can be found in [Feilkas et al., 2009b].

Further case studies are introduced if necessary to demonstrate particular concerns of system development, such as user interfaces or business information systems.

1.4 Outline

In this report we give an insight into an integrated approach towards model-based requirements engineering. We first define two exemplary overall artefact-based approaches. Afterwards, we introduce selected concepts and methods. The reminder of this report is organized as follows:

Chapter 2 is dedicated to the REMsES project. In close cooperation between industrial and academically partners, the recently completed REMsES project has developed a guideline to support requirements engineering processes in the automotive industry. The guideline enables the requirements engineers to cope with the increasing quantity, size, and complexity of such systems. This chapter presents the major results of the project, namely, the fundamental principles of the approach, the guideline itself, the tool support for REMsES-based processes, and the major results obtained from validation of the REMsES approach.

Chapter 3 is dedicated to the REMBIS approach. REMBIS provides a model-based reference approach for business information systems' analysis. It serves as an orientation for producing precise specification documents being conformant to the reference model. Based on this reference model, REMBIS defines a mechanism for a systematic and transparent customisation of the reference approach. The customisation approach offers the basic mechanisms for a customisation at organisational environments concerning process-integration and for a customisation at project environments. Regarding the latter, REMBIS defines how to systematically construct a RE process according to variable project influences.

Chapter 4 describes an approach to textual requirements patterns. The approach aims at improving the quality of textual requirements by reformulating them according to textual requirements patterns with a transparent underlying formal model. Hence the advantages of natural language (e.g. understandability, flexibility, structurability) are combined with the advantages of formal models (e.g. automotibility, analyses). The result of the proposed approach is a list of texutal requirements that follow a formal model - hence they are quite easy to transform in formats needed by more formal steps of the requirements and design phase.

Chapter 5 describes computational linguistics approaches to requirements engineering. Since natural language is the main presentation means in industrial requirements documents, typical requirements documents are incomplete and inconsistent. These deficits can be reduced with the aid of computational linguistics. The approaches presented in this chapter address two types of deficiencies: terminological inconsistencies and incomplete behavior descriptions. In both cases, deficiencies are detected by extraction of formal models from the text and inspection of the resulting models.

Chapter 6 describes an approach to formalizing functional requirements. Based on formulated and structured textual requirements, the requirements are formalized step by step. The core of the proposed approach is the notion of services. Services in this context are system functionalities that are visible to and perceivable by the user. Among services, there exists a couple of relations and interdependencies, that are of importance and should be taken into account while formalizing requirements. The notion of service hierarchy is another important notion to the proposed approach.

Chapter 7 describes an approach to modeling the usage environment of interactive systems. Since interactive systems are fundamentally constructed to interact with surrounding actors (human beings or further technical systems) in their usage environment, a thorough understanding of the usage environment is crucial for designing useful and usable systems. The proposed approach to environment modeling aims at building up a comprehensive model of the intended usage environment and defining the system's boundaries.

Chapter 8 presents an analysis method that describes human-machine systems by means of discrete mathematical models. The models are notated by a hierarchical automaton-based description technique. The main focus lies on the structured description of the interaction behavior and not on the graphical design of the user interface. The key concepts for the description of menu-driven user interfaces are combined within a conceptual model and integrated into the automaton-based description technique.

Chapter 9 presents a self-contained modeling approach aiming at the transition from requirements to design. It describes a subsystem across the three abstraction layers from user functions to technical architecture over logical architecture. An artifacts model for integrated requirements engineering and design gives an appropriate representation and documentation. The guiding method DeSyRe explains the system decomposition with help of a reference criteria catalogue, describes the transition from system requirements to subsystem requirements on the basis of the defined specification artifacts, and enables to extract subsystem specifications from the overall system specification so they can be handled independently (for assignment to suppliers or reuse).

Chapter 10 focuses on the importance of risk management activities at early development phase and proposes goal-driven modeling framework for managing software development risk even in early requirement engineering. The approach models the goals of the system under development, risk factors that obstruct the goals, and the treatment actions that influence the goals. This facilitates the assessment, management and monitoring of software development risk from early requirements engineering on.

Chapter 11 outlines a first summary of integrating requirements engineering approaches developed at TUM - Software & Systems Engineering (TUM I4). This draft draws out an overview of requirements engineering from the perspective of TUM I4 and states the current situtation in requirements engineering reasearch in the group. The big picture constitutes the basic for further integration efforts, expected in a near future.

2 Artefact-Based Requirements Engineering for Software-Intensive Embedded Systems

Birgit Penzenstadler
birgit.penzenstadler@in.tum.de

2.1 Motivation

The vision of the REMsES project was to develop an field-proven process guide for supporting requirements engineering processes in the automotive industry. Motivated by the three fundamental observations made in requirements engineering processes within the automotive industry, the REMsES guide aims at enabling engineers to cope with: (1) the growing quantity and size of siES in vehicles, (2) the increasing inter-system complexity of siES, and (3) the increasing intra-system complexity within large compound structures of siES.

2.2 REMsES Approach

REMsES is based on the following four principles[1] below. The consideration of these principles for the development of the REMsES guide supports in addressing the challenges described above.

P1: Differentiation of distinct Abstraction Layers in System Examination

P2: Differentiation of distinct Points of View

P3: Seamless Modelling

P4: Basing on Artefacts

The REMsES approach is artefact-based. This means that both the process model and the environment model are tightly connected to the artefact model as depicted in Fig. 2.1. The artefact model provides a basic structure for the definition of the artefacts, their assignment to abstraction layers and content categories, and the relations between the artefacts.

The process model defines the coarse-grained course of action (management view) and fine-grained task descriptions (development view). The progress of the general course of action is

[1]The term "principle" is used in the sense of an axiom or convention that provides the foundation for all action [Ghezzi et al., 1991].

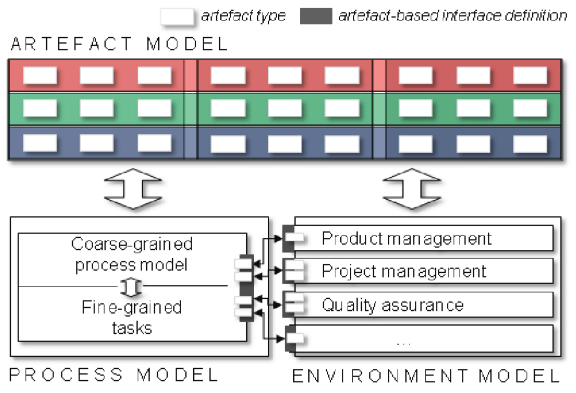


Fig. 2.1: Relation of artefact model, process model and environment

related to and therefore visible through the degree of completion of the artefact model. The interfaces between tasks or the accomplishment of a task are defined on the basis of the development and completion of certain artefacts.

The environment model defines the interfaces between the environment processes, for example product management, project management, development, ..., and the REMsES process. These interfaces can also be defined artefact-oriented by demanding for certain artefacts to be received, delivered or exchanged.

The REMsES Artefacts By choosing a particular content category, abstraction layer, and specification technique, the modeller obtains a detailed characterisation of the model to be created. However, the guidance for creating the model comes from three different sources, the descriptions of the content categories, the descriptions of the abstraction layers, and the descriptions of the specification techniques. Having to put together the guidance from the three sources may overwhelm the modeller. Thus, we provide predefined artefact descriptions for the possible choices of content category, abstraction layer, and specification technique. Fig. 2.2 shows an overview of the artefacts that result from applying the three abstraction layers to scenario modelling and architectural modelling.

The REMsES Guide The REMsES guide was realized as a hypertext-based system. On the highest level the REMsES guide consists of the more specifically *REMsES process guide* and the *REMsES demonstrator*. Using the example of a “Radio-Frequency-Warning-System”[2] the REMsES Demonstrator shows the various artefacts that will be created during a REMsES-guide-based requirements engineering process for a software-intensive embedded system. In the following, we focus on the second component of the REMsES guide, the REMsES process guide. The overall content structure of the REMsES process guide is shown in Fig. 2.3.

[2] The Radio-Frequency-Warning-System (RFW) is a vehicle system that detects the radio-signature of road signs. Depending on the type of detected road signs and the driving direction the system initiates specific actions, e.g. the system informs the driver about a legal speed limit.

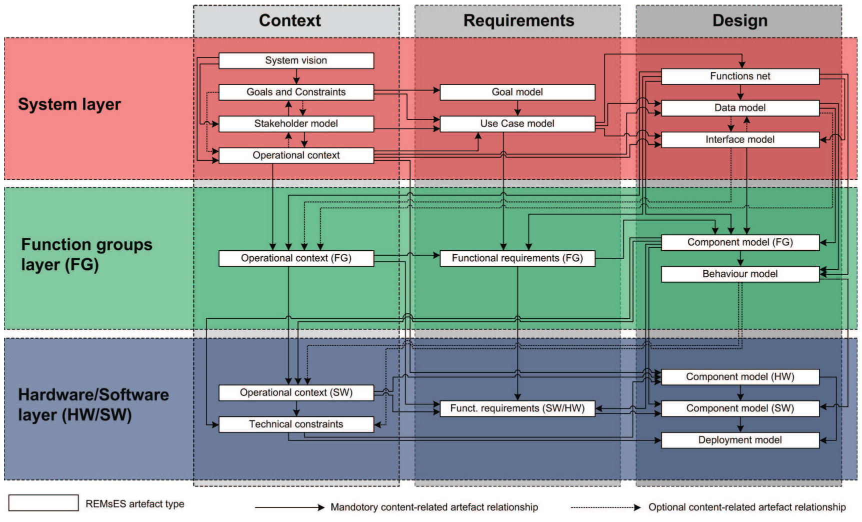


Fig. 2.2: Assignment of related artefacts to information categories.

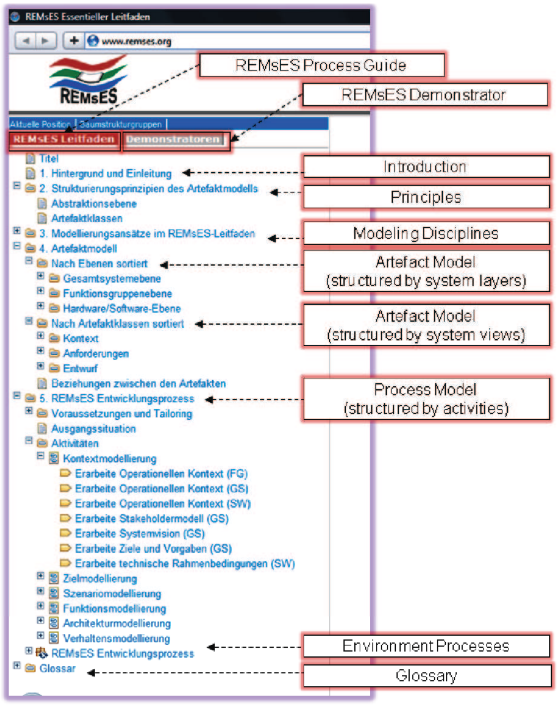
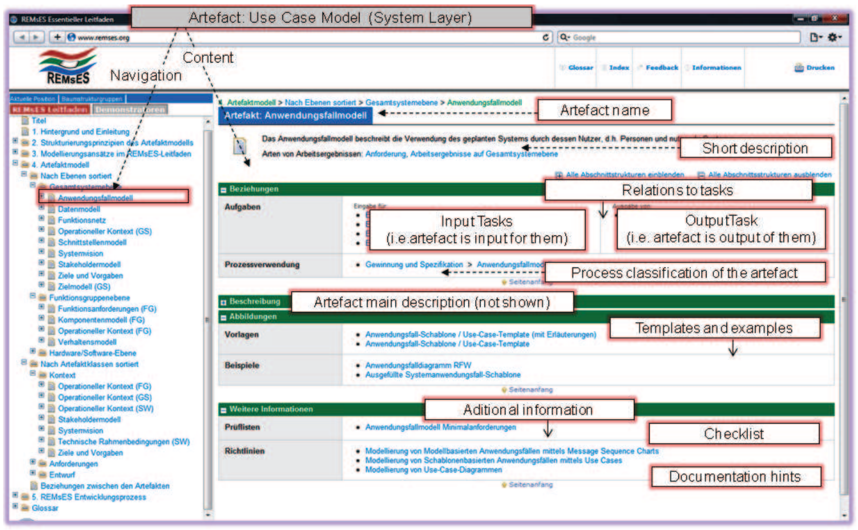

REMsES Essentieller Leitfaden
www.remses.org
REMsES
REMsES Process Guide
REMsES Demonstrator
Aktuelle Position | Baumstrukturgruppen |
REMSES Leitfaden
Demonstratoren
Titel
1. Hintergrund und Einleitung
2. Strukturierungsprinzipien des Artefaktmodells
Abstraktionsebene
Artefaktklassen
3. Modellierungsansätze im REMsES-Leitfaden
4. Artefaktmodell
Nach Ebenen sortiert
Gesamtsystemebene
Funktionsgruppenebene
Hardware/Software-Ebene
Nach Artefaktklassen sortiert
Kontext
Anforderungen
Entwurf
Beziehungen zwischen den Artefakten
5. REMsES Entwicklungsprozess
Voraussetzungen und Tailoring
Ausgangssituation
Aktivitäten
Kontextmodellierung
Erarbeite Operationellen Kontext (FG)
Erarbeite Operationellen Kontext (GS)
Erarbeite Operationellen Kontext (SW)
Erarbeite Stakeholdermodell (GS)
Erarbeite Systemvision (GS)
Erarbeite Ziele und Vorgaben (GS)
Erarbeite technische Rahmenbedingungen (SW)
Zielmodellierung
Szenariomodellierung
Funktionsmodellierung
Architekturmodellierung
Verhaltensmodellierung
REMsES Entwicklungsprozess
Glossar
Introduction
Principles
Modeling Disciplines
Artefact Model (structured by system layers)
Artefact Model (structured by system views)
Process Model (structured by activities)
Environment Processes
Glossary

Fig. 2.3: Structure of the REMsES navigation frame


Artefact: Use Case Model (System Layer)
Content
Navigation
Artefact name
Short description
Relations to tasks
Input Tasks (i.e.artefact is input for them)
OutputTask (i.e. artefact is output of them)
Process classification of the artefact
Artefact main description (not shown)
Templates and examples
Aditional information
Checklist
Documentation hints
Glossar
Index
Feedback
Informationen
Drucken

Artefaktmodell > Nach Ebenen sortiert > Gesamtsystemebene > Anwendungsfallmodell

Artefakt: Anwendungsfallmodell

Das Anwendungsfallmodell beschreibt die Verwendung des geplanten Systems durch dessen Nutzer, d.h. Personen und nu

Arten von Arbeitsergebnissen: Anforderung, Arbeitsergebnisse auf Gesamtsystemebene

Beziehungen

Aufgaben	Eingabe für: / Ausgabe von:
Prozessverwendung	• Gewinnung und Spezifikation > Anwendungsfallmod

Beschreibung

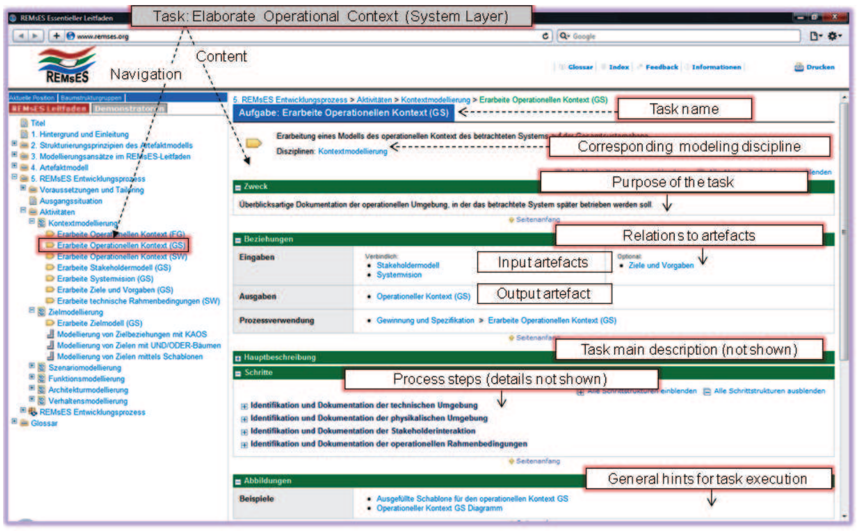
Abbildungen

Vorlagen	• Anwendungsfall-Schablone / Use-Case-Template (mit Erläuterungen) • Anwendungsfall-Schablone / Use-Case-Template
Beispiele	• Anwendungsfalldiagramm RFW • Ausgefüllte Systemanwendungsfall-Schablone

Weitere Informationen

Prüflisten	• Anwendungsfallmodell Minimalanforderungen
Richtlinien	• Modellierung von Modellbasierten Anwendungsfällen mittels Message Sequence Charts • Modellierung von Schablonenbasierten Anwendungsfällen mittels Use Cases • Modellierung von Use-Case-Diagrammen

Fig. 2.4: Example Artefact Description within the Guide


Task: Elaborate Operational Context (System Layer)
Content
Navigation
Task name
Corresponding modeling discipline
Purpose of the task
Relations to artefacts
Input artefacts
Output artefact
Task main description (not shown)
Process steps (details not shown)
General hints for task execution
Glossar
Index
Feedback
Informationen
Drucken

5. REMsES Entwicklungsprozess > Aktivitäten > Kontextmodellierung > Erarbeite Operationellen Kontext (GS)

Aufgabe: Erarbeite Operationellen Kontext (GS)

Erarbeitung eines Modells des operationellen Kontext des betrachteten Systems

Disziplinen: Kontextmodellierung

Zweck

Überblicksartige Dokumentation der operationellen Umgebung, in der das betrachtete System später betrieben werden soll

Beziehungen

Eingaben	Verbindlich: • Stakeholdermodell • Systemvision	Optional: • Ziele und Vorgaben
Ausgaben	• Operationeller Kontext (GS)	
Prozessverwendung	• Gewinnung und Spezifikation > Erarbeite Operationellen Kontext (GS)	

Hauptbeschreibung

Schritte

Alle Schrittstrukturen einblenden · Alle Schrittstrukturen ausblenden

- Identifikation und Dokumentation der technischen Umgebung
- Identifikation und Dokumentation der physikalischen Umgebung
- Identifikation und Dokumentation der Stakeholderinteraktion
- Identifikation und Dokumentation der operationellen Rahmenbedingungen

Abbildungen

Beispiele	• Ausgefüllte Schablone für den operationellen Kontext GS • Operationeller Kontext GS Diagramm

Fig. 2.5: Example Task Description within the Guide

The artefact descriptions in the REMsES process guide provide detailed information that is relevant for the documentation and quality assurance of the concerning artefact. The structure of an artefact description within the REMsES Guide is illustrated in Fig. 2.4. The REMsES process model consists of a coarse-grained and a fine-grained process model. The former defines general control flow dependencies within REMsES-guide-based requirements engineering processes. The latter model defines individual artefact-related tasks. Each of these tasks provide a structured description for supporting the systematic development of artefacts of the specific type. The structure of a task description within the REMsES Guide is illustrated in Fig. 2.5.

2.3 Case Studies

The REMsES guideline was reviewed in workshops with domain and process experts from the companies and empirically evaluated in case studies in experiments with students. The complete case study of the RFW system, the REMsES illustrator, is included in the guide described above.

2.4 Conclusion and Outlook

The REMsES project produced a freely available HTML guide[3] to requirements engineering for embedded systems. The results and insights of the project are now used as input for the SPES project.

[3]at `http://www.remses.org`

3 Artefact-Based Requirements Engineering for Business Information Systems

Daniel Méndez Fernández
mendezfe@in.tum.de

3.1 Motivation

The first step for companies towards RE excellence consists in the definition of a RE process at an *organisational level*. This includes the preparation of methods and techniques for a company-wide use. Furthermore, the process must be integrated into the development life cycle. This integration that establishes the association of RE-specific results with contents of further processes of the development life cycle is known as *process integration*.

Once the RE process has been defined and integrated for a company, it has to methodically guide through the elaboration of requirements at *project-level*. A RE process that is executed at the project-level must then support the precise definition of requirements. Preciseness in the definition of requirements includes aspects of structure, syntax and semantics regarding the content of the specification documents. Obviously, the process must include the basic concepts respecting the needs of individual application domains. Such a systematic RE approach achieves for requirements engineers *awareness* of constructing requirements specifications that are *conformant to domain-specifics* respecting *completeness* and *consistency* of the results. Completeness means that the results are described with all necessary elements, for example each use case has a unique identifier. Consistency concerns the consistency in the relations between different results, for example each use case description needs the explicit and separate description of an actor. Both completeness and consistency within and between the results are the basis for seamless modelling, thus, for *continuity* between the activities of the RE process and also between RE and further development processes.

Still, in practice the definition and the execution of RE processes is often complicated. It begins with a missing integration of RE aspects into available approaches that are specifically elaborated for the domain of business information systems. As a consequence of current approaches neglecting RE, companies often do not include RE aspects into their process models. They do neither define domain-specific methods and techniques for a systematic elaboration of requirements, nor integrate the results into the development life cycle. In addition, customer influences make the RE process highly variable at the project level. Exemplary influences are the availability of end users or the enforcement of used specification structures complicate the use of a standardised process. As a consequence engineers often act solution-biased in terms of adopting the solution rather then precisely specifying requirements since there is no systematic reference approach that can be taken as an orientation for producing precise results, the results remain incomplete and inconsistent. This, in turn, causes a disruption in the development life cycle, as the results being produced by one development activity are the necessary input for the next. Consequently, a quality downslide of the process, of the process' results and finally of the application being developed is observable.

Goals

Companies are facing the problem of standardising an efficient and effective RE process that fits into individual project environments. This includes (1) the definition and integration of a RE reference approach at organisational levels (2) which serves as an orientation at project levels, while (3) supporting a systematic and sustainable decision taking according to project-specific parameters.

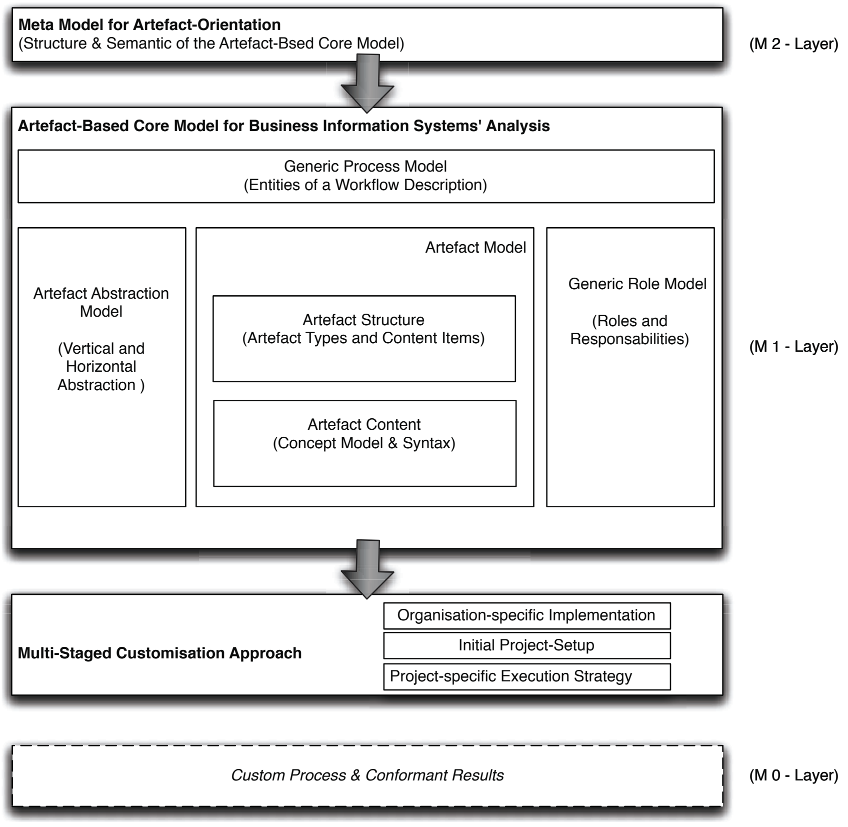
In order to tackle this problem, it demands for an approach that:

1. ensures consistency and completeness of the results to enable a seamless modelling and thereby continuity within the process. It has to precisely define the structure, the syntax, and semantics of the results specific for the chosen application domain.
2. can, at an organisational level, be deeply integrated into the development life cycle, i.e. be integrated into process models establishing relations to development-specific and management processes.
3. guides at project-level the reflection and decision making regarding project influences that arise before and during the execution of RE both affecting the construction of the results.

3.2 Approach

In order to achieve the defined goals, we contribute (1) a self-contained and artefact-based reference model for RE of business information systems and (2) a customisation approach that systematically adapts the reference model to organisational needs and to needs of individual project environments. A detailed description of the approach can be taken from [Mendez Fernandez and Kuhrmann, 2009].

However, artefact-orientation in general tackles the variability in designing a process as it concentrates on the results rather then emphasising a strict order of producing the results. The artefact-based paradigm defines the terminology and the content of the results (artefacts) as the backbone of project execution and therefore abstracts from concrete tools or methods. Artefacts offer a process-agnostic basis that can be customised, as artefacts are not restricted to process structures and specific techniques. If building a customisation approach upon an artefact model, instead of upon a process-based one, it offers the possibility of a process agnostic RE that is highly customisable. It can be customised to organisational environments and to project environments. The latter can be performed by allocating project parameters to the artefacts and guiding through artefact model according these influences, while still achieving both consistency and continuity.



Meta Model for Artefact-Orientation
(Structure & Semantic of the Artefact-Bsed Core Model)
(M 2 - Layer)
Artefact-Based Core Model for Business Information Systems' Analysis
Generic Process Model
(Entities of a Workflow Description)
Artefact Abstraction Model
(Vertical and Horizontal Abstraction)
Artefact Model
Artefact Structure
(Artefact Types and Content Items)
Artefact Content
(Concept Model & Syntax)
Generic Role Model
(Roles and Responsabilities)
(M 1 - Layer)
Multi-Staged Customisation Approach
Organisation-specific Implementation
Initial Project-Setup
Project-specific Execution Strategy
Custom Process & Conformant Results
(M 0 - Layer)

Fig. 3.1: Overview on the Approach

Figure 3.1 gives an overview on the approach. In general, we distinguish according to the Object Management Group (OMG) between three abstractions: (1) a meta model for the paradigm of artefact orientation, (2) the artefact-based core model that serves as a reference at project-level including the basic concepts of the application domain of business information systems and finally (3) the individual process and the results that are systematically constructed at the project-level using the customisation approach for guiding through the reference model.

3.2.1 Meta Model for Artefact Orientation

The meta model describes the language for the artefact-based paradigm and aims at precisely defining structure and semantics of the paradigm. The precision in the structure is necessary for enabling the systematic customisation of the core model to organisational environments and to individual project environments.

3.2.2 Artefact-Based Core Model

The core model contains concrete elements of the artefact-based infrastructure that serve as a reference model for individual projects. The model results from a co-operation between the Technische Universität München and Capgemini sd&m, a major German company for custom software development. The model consists of an artefact model, an artefact abstraction model that defines abstraction within the artefact model, a generic role model, and a process model both coupled to the artefact model. The reference model includes therefore all domain-specific entities that are necessary to define an artefact-based process for business information systems.

Artefact Abstraction Model. The artefact abstraction model defines horizontal levels of abstraction respecting the stages of refinement of different RE-specific results and modelling views that structure each level according to environment, behaviour, structure and information. The levels and the views serve as a basis for systematic cross-entries into the development project and traceability among consistently defined results. The levels are classified into two major areas concerning the description of the structure of an organisation and of the business, being compliant to the description of requirements towards the underlying IT infrastructure.

The *organisation's context* defines the high-level, long-term steering principles and objectives of a company. The *business process hierarchy* describes the organisation's structure, thereby a taxonomy of the main offered business activities that fit into the organisation's context. Derived from this structural view, the *business process logic* emphasises the internal realisation of the activities in terms of a business process model defining a workflow description including interchanged information and participating roles. The *information system service hierarchy* defines the external behaviour of single systems in terms of defining what parts of the business process model shall be supported by system's functionalities. It defines user-visible characteristics of the systems as a whole by such means of information system service descriptions or a use case model. The *information system's constraints* derive single quantified requirements that address the system in its applications, its architecture and its environment.

Finally to define the scope in-between the different levels that address the business and system descriptions, we make use of visions, e.g. the system vision that defines the system's context and summarises all the relevant information system services in order to steer all stakeholders into a common direction, before initiating analysis activities in full that address subsequent levels.

Artefact Model. The artefact model builds the backbone and provides a combination of both the structure of the specification documents and the content defined by a concept model. The structure addresses the aspects of hierarchically ordered documents and contents being produced during development tasks in which single content items serve as containers for the concepts, respectively concept models. The definition of this taxonomy is inspired by previous work of the RE group, namely by REM [Geisberger et al., 2006]. The concept models (the artefacts' content) extend the taxonomy with those details of a system under consideration dealt with during the development process [Schätz et al., 2002] and reflected in the elements and their relations found in the description techniques, respectively notions [Schätz, 2008]. For instance, it defines the exact contents of use cases and dependencies to further concepts of RE and other development phases. Furthermore, we define for each concept in the model possible notions by analysing syntactical possibilities for representing the artefacts. Finally, we define the interfaces of single artefacts to further processes of the development life cycle. For instance, by defining what artefacts serve as an input for specific project management activities like the definition of contractual agreements.

Figure 3.2 illustrates an excerpt of the artefact model. The left side of the figure depicts the structural view, the right side a view onto the concept model. The concept model is not discussed in detail, but illustrates how it serves as a basis for the definition of terminology and consistency rules of all produced entities of a project.

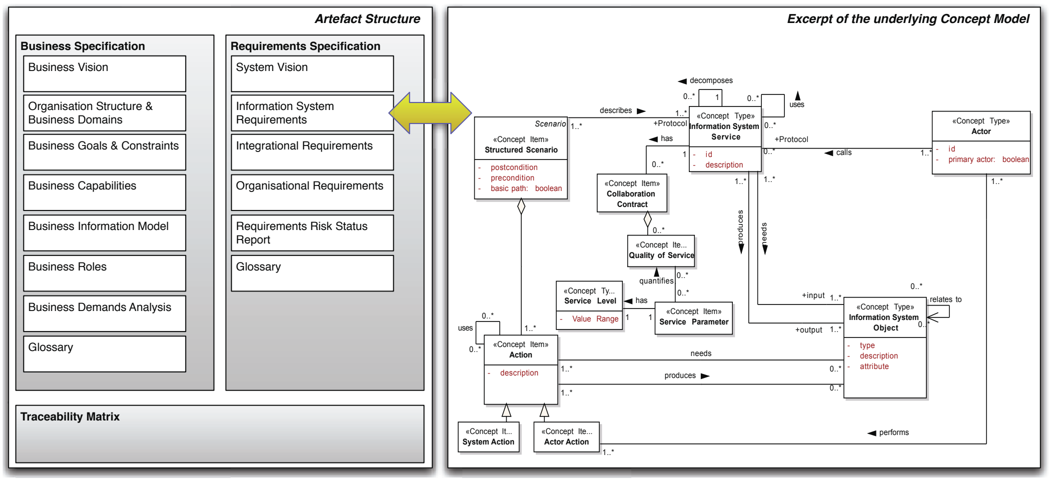

Artefact Structure
Business Specification
Business Vision
Organisation Structure & Business Domains
Business Goals & Constraints
Business Capabilities
Business Information Model
Business Roles
Business Demands Analysis
Glossary
Requirements Specification
System Vision
Information System Requirements
Integrational Requirements
Organisational Requirements
Requirements Risk Status Report
Glossary
Traceability Matrix
Excerpt of the underlying Concept Model
decomposes
uses
describes
Scenario
«Concept Item» Structured Scenario
postcondition
precondition
basic path: boolean
+Protocol
«Concept Type» Information System Service
id
description
has
calls
«Concept Type» Actor
id
primary actor: boolean
«Concept Item» Collaboration Contract
«Concept Ite... Quality of Service
quantifies
produces
needs
«Concept Ty... Service Level
Value Range
«Concept Item» Service Parameter
+input
+output
relates to
«Concept Type» Information System Object
type
description
attribute
«Concept Item» Action
description
needs
produces
«Concept It... System Action
«Concept Ite... Actor Action
performs
0..*
1
1..*

Fig. 3.2: Excerpt of the Artefact Model

By building the bridge between domain-specific principles of model-based development and structural aspects into which the models are embedded we define an integrated model. It ensures awareness of engineers towards the production of consistent and complete results and thereby a continuous workflow in the process. To support this awareness of produced artefacts being conformant to the artefact model, we define in addition conformance constraints in terms of conditions for completeness and consistency, both by the use of a logic-based formalism.

We enable a process-agnostic and customisable basis. It can be integrated to individual process models at organisational levels. Furthermore, it can be customised to the needs of projects, as we can define for single concepts the project parameters that take effect and resulting decisions to take, both respecting their construction (see also the multi-staged customisation approach).

Role Model and Process Model. The core model defines finally roles and a process model. The role model defines in particular the roles (and responsibilities) that directly participate in the execution of the process regarding the elaboration of the artefacts or indirectly contribute to this elaboration. The process model defines a framework including all the entities that are necessary to define a workflow description. This workflow description contains the definition of methods that are performed in order to produce, modify and use artefacts [Brinkkemper, 1996, Braun et al., 2005]. It defines phases, activities, tasks (respectively methods), and milestones, all workflow entities coupled to elements of the artefact model.

3.2.3 Multi-Staged Customisation Approach

The customisation approach is conducted over two basic stages. The first stage concerns the organisation-specific implementation, i.e. process integration. The latter stage concern the customisation to project environments

Organisation-Specific Implementation. The approach defines the mechanism for customising the core model according to organisational environments. We show how to integrate the model into existing process models. The only prerequisite for such an integration is a process model at organisational level that is based on a meta model such as it is the case for the V-Modell XT, a German standard for project management.

Customisation to Project Environments. Regarding a customisation at project level, we refer to an experience-based customisation. First, we define how to set-up of a project *before* project execution. This includes aspects like the definition of roles or the elaboration of initial artefacts based on given specifications (like given during initial bidding procedures). Second, we define how to elaborate the artefacts *during* project execution. For this purpose, projects are characterised by selected project parameters. They are in particular characterised by the parameters' effects onto the ability and necessity of constructing the artefacts to which the parameters are coupled. The effects are based in turn on the values of chosen measurements for the parameters reflecting the experiences of already performed projects. For example, by defining how the availability of end users affected the ability of defining a use case model and what decision have been taken.

We define therefore during project execution the mechanism for customisation by giving guidance on what project parameters influence the necessity and ability of constructing the concepts, while still respecting domain-specific structures within the results. Hence, we define when to reflect on what project parameters, what decisions to take, and what impact these decisions have on the construction of further concepts. As the customisation is based on the concept model, it still ensures consistency and completeness of and between the results. As the process model is coupled to the concept model model, it ensures continuity within the process.

This finally enables a systematic customisation that leads to conformant results at the project level and thereby creating overall awareness of a domain-specific RE process.

3.3 Case Studies

A case study comprises an evaluation of the approach concerning the applicability of the customisation steps, specifically for the application domain of business information systems. Consequently, it is performed over two stages:

1. Organisation-specific integration, i.e. comprehensive process integration that is performed with the V-Modell XT (see also [Mendez Fernandez and Kuhrmann, 2009]).

2. Customisation to Project Environment in which the comprehensive approach (including the core model) is applied to a real life project. This evaluation is currently in progress by

performing an action research study within a project with a duration of one year.

3.4 Conclusion and Outlook

The definition of the artefact-based model for the application domain of business information systems offers a customisable approach that resists volatile process environments. The artefact model itself abstracts from methods provided by several approaches of the RE group, such as [Thurner, 2004] regarding business processes or [Broy, 2003, Broy, 2005] regarding the description of services. Therefore, the validity of the artefact model and its integration is ensured in terms of being correct and integrated with concepts and terms of the RE group of the Technische Universität München.

The customisation approach, however, defines a mechanism to customise the model to organisational needs and to the needs of individual project environments. Regarding the latter we enable:

1. A systematic RE execution strategy as the process is systematically constructed within a predefined framework according to variable individual parameters
2. Transparency as decisions can be comprehensibly taken and remain reproducible

Finally, we are currently performing a field study for the elaboration of exemplary project parameters. This enables the initial construction of the framework for decision taking.

4 Textual Requirement Patterns

Andreas Fleischmann
andreas.fleischmann@in.tum.de

4.1 Motivation

One problem of requirement engineering is to find the right way to document requirements: a documentation format that is precise, that supports analysis and quality checks (consistency, unambiguity, correctness, completeness), that allows a smooth transition to the design, and that still is well understandable by all the stakeholders. While several models techniques improve the precision and automatization of requirements, textual requirements are still the best communication means for all the stakeholders involved.

Hence, the approach summarized in this chapter develops a *precise* specification format of the requirements by the help of pre-defined *textual* patterns and a method to define the vocabulary of those requirements, and hence supports a seamless transition between the informal requirements engineering phase and the formal design phase (a full description of this approach can be found there [Fleischmann, 2008]).

4.2 The Methodology

In the presented approach, the informally and textually given requirements are analyzed, categorized, formulated, and structured. This is done in four steps:

1. Establishing an *initial template*.
2. *Categorizing* the requirements.
3. Defining the *logical interface*.
4. *Formulating* the requirements.

In the following paragraphs, these four steps are described.

Establishing an initial template It is assumed that the requirements are gathered by an elicitation subprocess, which delivers requirements as informal text. In order to be further processed, these informally given requirements have to satisfy two initial properties: each requirement must have a unique number, and each requirement must describe only one aspect of the system's behaviour. This prerequisite is achieved by putting each requirement into a initial template which consists of the following fields:

- **Id** (mandatory): a unique identifier of the requirement, e. g. a number.
- **Name** (mandatory): some words that briefly label the requirement.
- **Description** (mandatory): the content of the requirement; initially an informally given text consisting of one or more sentences.
- **Comment** (optional): a further explanation of the requirement, which does not contain a crucial requirement content, but instead gives a reason for the requirement, or elaborates the meaning of the requirement.

These four fields are the fields needed by the formulation part. Of course, a requirements template can contain a lot more fields (e. g. author, date, priority); that doesn't affect the presented approach.

Categorizing the requirements Since requirements are very heterogenous, it is not possible to structure and formulate all of them in the same way. Therefore the requirements are categorized into the following four homogeneous categories:

- **Business Requirements**: Such requirements do not describe what the system is doing; instead they describe *why* a company wants to develop a system, what the company wants to achieve with the product. Often business requirements appear not as individual requirements, but as rationale for user requirements. Typical business requirements are, for example: "With this new product, the market share should be increased by 10 per cent" or "because this is a precondition for the product's usage in the European market."
- **User Requirements**: Such requirements describe the system's properties and behaviour as it is perceived by the user of the overall system (e. g. the driver of the car).
- **System Requirements**: Such requirements describe the system's properties and behaviour as a part of the overall system that communicates with actuators, sensors, interfaces, channels, and other embedded systems.
- **Process Requirements**: Such requirements do not describe what the system is doing; instead they describe *how* the system is supposed to be developed, e. g. the application of standards, laws, and development techniques (e. g. testing and verification directives).

For each category, specific formulation rules can be defined. For the category of user requirements, those rules have been defined (see next paragraphs). Hence, in the following, only *functional user requirements* will be further processed by applying those rules. All other requirements are also important, but their formalization is a task for future work; until then, those requirements will just stay untransformed in the initial template.

Defining the logical interface An informally given textual requirement consists of a set of words, which can be divided into two categories:

- Content words (**actions**): these are system-specific words (e. g. "driver presses on/off button" or "acc terminates"). The set of all content words forms the **logical interface** of the system: with what means does the system react, and what are the relevant events a system reacts to.
- Relation words (**keywords**): these are domain-independent words (e. g. "if", "then"). These words form relationships between the content words, and therefore define the **behaviour and properties** of the system.

In the first step, the requirements are analyzed, and the actions (content words) are identified and extracted. The extracted actions are put into tables with the following fields:

- **Name**: the name of an action is identical with the action itself; it must be unique. Usually, in a specification document there are different ways used to express the same meaning, e. g. "system decelerates" and "system brakes" might mean the same. Whenever this is the case, one expression has to be chosen and all others discarded. As a result, for each meaning there is exactly one expression. While resolving these multiple expressions, the requirements engineer often has to ask, if two expressions really mean the same thing: these questions often lead to a better understanding of the logical interface of the system.
- **Input/output**: it has to be defined, whether an action describes an event, to which the system reacts (input), or means, with which the system reacts (output).
- **Type**: an action can be one of the following types:
  - *Event*: an event describes a moment, in which the system realizes something or does something; an event has no duration. Examples are: a button gets pressed, another car appears within sensor range.
  - *State*: a state describes a time period, in which something is. Examples are: a button is pressed, another car is within sensor range.
  - *Activity*: an activity describes a time period, in which the system does something. Examples are: acc accelerates, acc warns driver.

  It is crucial to decide what type a content word is; in today's practice this distinction often is not clear. The clarification of the type is a big factor in better understanding the logical interface of a system.
- **State space**: states and events are structured within a table by sorting them to an appropriate state space. For example, the states "acc is active" and "acc is not active" belong to the same state space. Within a state space, certain rules apply:
  - all states of a state space are disjunct.
  - the system must be in exactly one state of a state space all the time (hence, a state space must be complete, so that this rule always is true).
- **Variables**: within actions there sometimes are variables that they reference; e. g. the action "driver-increases-target-speed" references the variable "target-speed". In order to better understand dependencies between actions, which can be recognized by their reference to the same variables, such variables are explicitly noted.

Formulating the requirements In the previous step, the system-specific content words have been extracted from the requirements sentences and a logical interface has been defined. Now, in the last step of this phase, the requirements themselves are *formulated*. There, two rules apply:

- Only the standardized words from the logical interface can be used.
- Sentences can be formed according to requirements patterns only.

The following requirements patterns are defined:

- Reaction Behaviour Patterns
  - Direct Reaction of System: IF an event occurs WHILE a situation (optional) THEN the system reacts with another event
  - Direct Change of System: IF an event occurs WHILE a situation (optional) THEN a system state changes
  - Prohibition of Reaction: IF an event occurs WHILE a situation (optional) THEN an activity or event is forbidden
- Situation Behaviour Patterns
  - Situation Behaviour: WHILE a situation THEN an activity is performed
  - Behaviour Restrictions: WHILE a situation THEN an activity is restricted
  - Prohibition of Behaviour: WHILE A situation THEN an activity or event is forbidden
- Invariant Patterns are directly formulated within the action tables.

The requirements are *structured* according to their preconditions (that is the list of situations belonging to a requirement). This structure is used as a starting point for the service tree in Section 6. The logical requirements interface is extended to the logical syntax interface. And the formulated requirements are the foundation for defining the behaviour of the services.

4.3 Conclusion and Outlook

The approach works well with functional requirements and has proven its usefulness in several case studies. The formulated requirements (text surface with an underlying model) are the perfect starting point for more formal requirements or design methods (pure model). However, to be applicable in real-scale projects, it must be extended to nonfunctional requirements as well, because they are a very important part of the specification.

5 Computational Linguistics in Requirements Engineering

Leonid Kof
kof@in.tum.de

5.1 Motivation

At the beginning of every software project, some kind of requirements document is usually written. The majority of these documents are written in natural language, as the survey by Mich et al. shows [Mich et al., 2004]. This results in requirements documents that are imprecise, incomplete, and inconsistent, because precision, completeness and consistency are extremely difficult to achieve using mere natural language as the main presentation means.

The authors of requirements documents are not always aware of these document defects. Even documents that are precise from the human point of view can omit some facts relevant for behavior specification. According to Boehm [Boehm, 1981], the later an error is found, the more expensive its correction. Thus, it is one of the goals of requirements analysis, to find and to correct the defects of requirements documents. Our goal is to detect the defects by means of computational linguistics.

5.2 Document Analysis

5.2.1 Ontology Extraction from Requirements Documents

Requirements Engineering involves many stakeholders and includes not only technical but also sociological and psychological activities. Even when all the stakeholders come to a consensus, the produced requirements are rather informal. In the early project phases the functionality of the prospective software is not yet understood in the precision necessary for formalization, which makes requirements formalization not only a refinement, but also a learning process.

The first step of this learning process should be a definition of a common project vocabulary, either as a glossary or as an ontology. To establish such a common vocabulary, we propose an approach to ontology extraction from textual documents. An ontology consists of a set of terms and relations between these terms. As compared to a glossary, a domain-specific ontology gives a more explicit definition of terms and relations between them. When the ontology is extracted, a domain expert validates it. The validated ontology becomes both *the* common language for all the project stakeholders and a valuable resource for later development steps.

In computer science an ontology consists of three parts: a list of concepts, a classification of these concepts ("is-a" relation, taxonomy), and a set of non-taxonomic relations. An ontology can be extracted from a requirements documents in four steps, as presented in [Kof, 2005]:

- Every sentence is parsed, the subject and the objects of every sentence are extracted.
- Each subject and each object is clustered according to grammatical context in which it occurs.
- Every pair of overlapping clusters is joined to a larger cluster, representing more general concepts. "is-a" relation (taxonomy) is derived from the cluster hierarchy: Every cluster represents some concept, the larger the cluster, the more general the concept. Cluster inclusion means the "is-a" relation between the corresponding concepts.
- A non-taxonomic relation is assumed for every two concepts that often co-occur in the same sentence.

The third and the fourth steps are *interactive* and give the requirements analyst feedback on terminology inconsistencies. It is important to eliminate these inconsistencies *before* they find their way into the ontology (the requirements engineering process goes back to the step of requirements elicitation, negotiation, and writing). This interactive process of ontology extraction and document correction has the invaluable side-effect of validating the terminology that will be used in later project phases.

5.2.2 Extraction of Behavior Models from Requirements Documents

Most existing and really used formalisms for behaviour modeling are based either on interaction sequences or on automata, cf. [Kof and Schätz, 2003]. Thus, when translating textual specifications to behaviour models we focused on two formalisms: Message Sequence Charts (MSCs), as representatives for interaction sequences, and automata. In our work [Kof, 2007a, Kof, 2007b, Kof, 2008, Kof, 2009b] we developed approaches extracting behaviour specifications (MSCs and automata) from requirements documents even in the presence of certain defects. Both algorithms rely on Part-of-Speech (POS) tagging of every sentence. POS tagging marks every word of the sentence as a substantive, verb, adjective, etc. Available POS taggers have the precision of about 97% [Curran et al., 2006], which makes them unlikely to become an error source on its own.

The first step of both translation algorithms is the identification of states (in order to construct automata) or communicating objects (in order to construct MSCs). Identification of states and objects is performed by seeking predefined keywords followed or preceded by special sequences of POS tags, cf. [Kof, 2009a]. After the identification of states or communicating objects, the algorithms work in different ways:

- The algorithm for text-to-automaton translation decides for every sentence, whether this sentence represents a state transition, a context setting (like "the following paragraph specifies state transitions from the state XY"), or is irrelevant for the text-to-automaton translation. The decision depends on the occurrence of one of the previously extracted states in the sentence, and on the exact place where the state occurs, cf. [Kof, 2009b]. After this decision, the algorithm translates every sentence marked as "state transition" to a transition of the automaton. In real requirements documents, sentences often specify the resulting state of the transition only, but not the initial state. In this case, the algorithm infers the initial state of the transition from the context.

- The algorithm for text-to-MSC translation decides for every sentence, whether the sentence should be translated to a message or to an assertion. For example, passive sentences like "The system is turned on" are translated to assertions. For sentences translated to messages, the algorithm tries to identify the message sender and receiver, by searching the occurrences of previously extracted communicating objects in the sentence. If the message sender or receiver cannot be identified, the algorithm infers them by analyzing the message flow. Details of the message flow analysis can be found in [Kof, 2007a].

Both approaches can be used to detect document deficiencies: whenever inference becomes necessary, it signifies the presence of specification incompleteness.

5.3 Case Studies

5.3.1 Ontology Extraction

Applied to the Smart Key Entry Systems, the ontology extraction produces the following results:

Extracted terms: There are only three verbs (except for "be") that are used frequently enough to entail a cluster of concepts. The resulting extracted concepts are the following:

Different types of sensing zones: subjects of "is created": sensing zone to detect smart key, sensing zone to detect smart key in outside of cabin, detection zone, sensing zone to detect smart key in outside of trunk, sensing zone to detect smart key in inside of cabin, sensing zone to detect smart key in inside trunk, sensing zone, sensing zone of locked door, sensing zone to detect smart key in inside/outside of cabin, sensing zone to detect smart key in inside/outside of trunk, sensing zone to detect smart key

Different functions: subjects of "is available": power window close, switching door unlock mode, engine start

Different functions: subjects of "by pressing": unlock only driver door, unlock all doors, lock doors, unlock doors

Elements of user interface: objects of "by pressing": driver door handle switch, every door handle switches except driver door, outside door handle switch

The concepts clusters are disjoint, which makes taxonomy building impossible. However, we can still find non-hierarchical relations between the extracted concepts: for example, the function "unlock only driver door" and the interface element "driver door handle switch" occur in the same sentence, which signifies a non-taxonomic relation. When a human analyst is given such a suggestion for a non-taxonomic relation, the relation becomes obvious: “interface element causes function execution”.

Apart from ontology extraction, the approach can be used to examine terminology consistency: for example, different types of sensing zones include both “detection zone” and “sensing zone to detect smart key”. It is a question to the document author, whether these are just two different names for the same concept or really two different concepts with intentionally similar names. Such examination of resulting concept clusters can unify terminology and contribute to a common unified project language.

5.3.2 Extraction of Behavior Models

Smart Key Entry Systems does not contain explicit behavior specifications: it just says that some sensing zone should be created or some function should be called. It does not specify, however, how a sensing zone is created or how a particular function is performed. For this reason, behavior extraction was demonstrated on other specifications: The Instrument Cluster [Buhr et al., 2004] and The Steam Boiler Specification [Abrial et al., 1996]. The results of the case studies are too bulky to be presented here, they can be found in [Kof, 2007a, Kof, 2007b, Kof, 2008, Kof, 2009a, Kof, 2009b].

5.4 Conclusion and Outlook

The presented approaches automate parts of the step from requirements documents to design. Despite minimal assumptions about the structure of the sentences to be translated, the approaches are effective, which was shown in case studies. When modeling system behavior, the translation of texts to design imitates the way how human analysts would model the discourse context. This context model is then applied to infer information not explicitly stated in the behavior specification. These approaches have high potential for industrial application, as the majority of industrial requirements documents are written in natural language.

6 Stepwise Formalization of Functional Requirements

Sabine Rittmann (Edited by Wassiou Sitou)
{sabine.rittmann|wassiou.sitou}@gmail.com

6.1 Motivation

Requirements engineering aims at specifying "what" the system-to-be is supposed to do. Thus, the requirements specification contains the functionality that the system shall provide, constraints concerning the distribution and the technical realization, etc. The specification of requirements is not a straight forward task: requirements have to be specified precisely (without ambiguities), consistently and as far as possible totally. Also, the requirements have to be validated, e.g. by simulation, to answer the question: "Does the requirements specification describe what I want?" There exists a natural gap between the informal requirements engineering phase (dealing with natural language) and the formal design phase (dealing with models).

With the following approach we address these challenges. Thereby we focus on the functional requirements for multi-functional systems. Multi-functional systems are characterized by a high degree of dependencies and interactions between functional entities. This kind of systems is typical for the automotive domain where the functionality is often provided by the interplay of several ECUs. The dependencies have to be captured and specified already in the requirements engineering phase. The main objective of the proposed approach is to facilitate a seamless transition between the informal requirements engineering phase and the formal design phase. The application of the proposed approach results in a formal model of the usage behavior that can for example be checked for totality and simulated in order to validate the requirements. The ideas presented here are explained in detail in [Rittmann, 2008]

6.2 Stepwise Formalization of Functional Requirements

Our approach to formalizing functional requirements is organized into interrelated steps. In the following, we describe the consisting steps of the proposed approach.

Identification of atomic services As the overall system functionality can be quite comprehensive, we structure it by *services*. A service is a *partial behavior* which relates system inputs to system outputs. Thus, a service describes a piece of usage behavior which can be observed at the system boundaries. The system functionality is comprised of single services which collaboratively establish the overall functionality. By services, the usage behavior of a system is structured. First, we identify the so-called *atomic services* which are the "smallest" services which are visible to and can by accessed by the user. A service corresponds to a use case which describes how to use a system for a specific purpose by means of interaction patterns. Examples for atomic services are "open the power window", "turn on the radio", or "move the seat toward the steering wheel". Usually, there is no 1:1 mapping between functional requirements and services and that it requires genuine design work to identify the services out of the requirements. Some of the atomic services need to operate on peristent data. For example, the speed control of the ACC functionality needs to save the desired speed at which the car is supposed to go. Therefore, we also informally specify this data. As services might operate on the same data[1] we list all persistent data in a repository. The informal service specifications then refer to entries of this list. The atomic services and the persistent data are only specified *informally* at this stage of the methodology. We suggest to use tables for their specification (see Tables 6.1 and 6.2).

Tab. 6.1: Specification of atomic services

Req	**Service name**	**Abbreviation**	**Textual service description**	**Data ref.**
...	...	...	...	...

Tab. 6.2: Specification of persistent data

Abbreviation	**Name**	**Description**
...	...	...

Logical syntactic system interface As mentioned above, services define pieces of the usage behavior by mapping system inputs to system outputs. The inputs and output were already identified (see Section 4.2). At this place, we specify the inputs and outputs more precisely by adding more information. Although the inputs and output of the system are still considered at a logical level (e.g. we still abstract from technical signals and refer to the meaning instead), we assign data types and data ranges to the inputs and outputs, respectively. During modeling the functionality formally (see Section 6.2) not all the previously inputs and outputs might be needed. Furthermore, several inputs might be integrated to another input. Therefore, in this step we only specify the inputs and outputs which we actually need within the formal models. As notational technique, we suggest to use tables in order to specify the logical inputs and outputs, respectively. Table 6.3 shows an example structure for the specification. For each input/output, the requirements are listed which motivate the introduction of the input/output. Furthermore a name and an abbreviation are given. A data type is also introduced. However, the data type does not have to be the data type which is needed during the implementation. Rather should it be chosen in a way which is appropriate for modeling the functionality on the logical level.

[1]In that case one might need to solve conflicts if more than one services want to write the same data concurrently.

Tab. 6.3: Specification of logical (input and output) actions

Reqs	Abbreviation	Name/discription	Data type
...	...	...	...

Identification of service relationships So far, we have captured the single atomic services in a modular fashion. In this step, we relate the services with each other. As mentioned above, multi-functional systems are intricate to develop as they are – per definition – characterized by a high degree of interaction between functionalities. Therefore, we explicitly capture the various dependencies, i.e. the *service relationships*, between them. As from our perspective, the system is a black box behavior, only relationships which are observable at the system boundaries, are taken into consideration. For example, the interruption of a service, but not the call of a sub functionality. First, we structure the overall usage behavior *hierarchically* by so-called *vertical service relationships* (also called sub service relationships). Graphically the result is a tree in which the nodes are services and the edges represent the sub service relationship. The leaves of the tree are the atomic services identified in Section 6.2; the root is the overall system behavior. Note that our notion of service is scalable. A partial behavior can both be a small, atomic service and a more comprehensive functionality. For example, the ACC functionality is a service again which is comprised of the sub services Follow-up Control and Constant Speed Control. We call this ordering of the system services according to the sub service relationship the *service hierarchy*. Next, we enrich the service hierarchy by the dependencies between the services, the so-called *horizontal service relationships*. For example, either the Follow-up Control or the Constant Speed Control is executed. Thus we have an XOR relationship between these (hierarchically decomposed) services. Each horizontal service relationship is informally specified. For example, the specification of the XOR relationship contains the information that the Follow-up Control is executed if no vehicle is detected and that otherwise the Constant Speed Control is executed. The result is a table containing the informal specification of the horizontal service specifications and the so-called *service graph* which is the service hierarchy plus the dependencies.

Formal specification of atomic services In the previous steps we have modeled the usage behavior informally. In this step, we get formal. We specify the atomic services (i.e. the leaves of the service hierarchy/graph) formally. To that end, we make use of AutoFOCUS Mode Diagrams. Mode Diagrams specify the system behavior depending on the mode (state) the system is in. Mode diagrams consist of modes (states) and transitions between these modes. Transitions are labeled by predicates overn the system inputs and outputs and define mode transitions. For each mode, either a Mode Diagram is given in turn (i.e. the modes can be decomposed hierarchically), or the behavior is described by means of a data flow network. Entities in Mode Diagrams can make use of modes again.

Combination of services on basis of the service relationships So far we have only modeled the atomic services of the service hierarchy/graph formally. In this step, we combine the atomic services to more comprehensive services until the overall usage behavior is obtained. Doing this we make use of a bottom up approach concerning our service hierarchy. During the combination process, the horizontal services relationships (i.e. the dependencies between the services) have to be taken into account. For example, if there exists an XOR relationship between services, they can be combined using modes. The XOR related behavior is specified in a (separate) mode, respectively. The mode transitions are given by the information which is specified for the XOR relationship.

6.3 Application to the ACC Case Study

The proposed methodology has been applied to the ACC case study. In the following we present the results of applying the methodology. Thereby, we briefly comment for each step the achieved results.

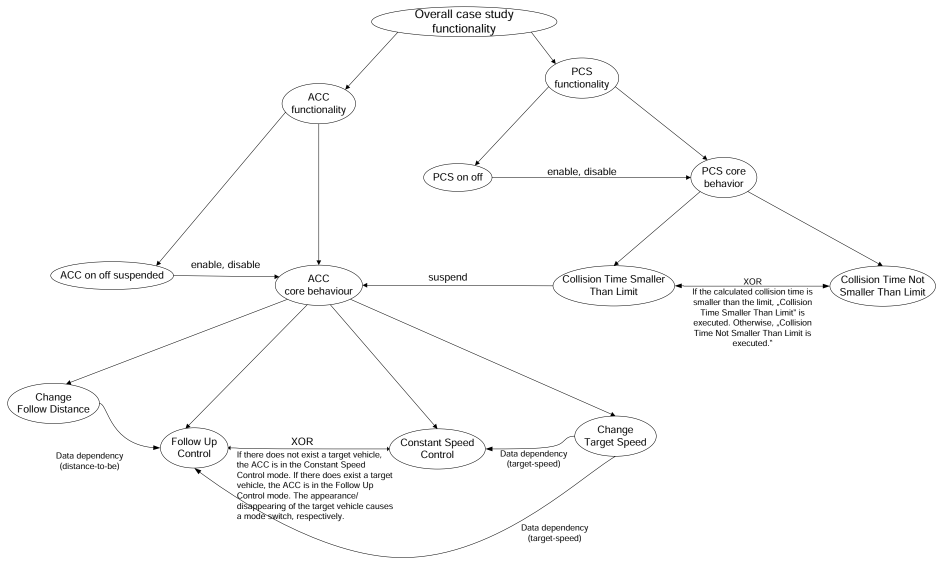
Identification of atomic services For the case study, we obtain examplatory atomic services. The data that has to be exchanged between the services is shown in Table 6.4.

Tab. 6.4: Specification of persistent data

Abbreviation	**Name**	**Description**
distance-to-be	distance mode	The driver can chose the distance mode (“short”, “medium”, “long”) for the Follow Up Control.
target-speed	Target speed	The target speed as chosen by the driver.

Logical syntactic interface In this step we list the logical inputs and outputs as needed for the formal modeling of the usage behavior. Note that the process of identifying the logical inputs and outputs is iterative and intermingled with the modeling of the system behavior.

Identification of service relationships In this step, we identify the service relationships of our case study. To that end, we first structure the services hierarchically (service hierarchy) by vertical relationships and then enrich the service hierarchy by horizontal dependencies. The resulting service graph is shown in Figure 6.1. The overall usage behavior is comprised of the “ACC functionality” and the “PCS functionality”. The “ACC functionality” can be again decomposed into the functionality to turn it on and off the core behavior. The core behavior is made up of the atomic services “Follow Up Control”, “Constant Speed Control”, “Change Follow Distance”, and “Change Target Speed”. We introduced the service “ACC core behavior” to be able to



If the calculated collision time is smaller than the limit, „Collision Time Smaller Than Limit" is executed. Otherwise, „Collision Time Not Smaller Than Limit is executed."

If there does not exist a target vehicle, the ACC is in the Constant Speed Control mode. If there does exist a target vehicle, the ACC is in the Follow Up Control mode. The appearance/ disappearing of the target vehicle causes a mode switch, respectively.

Fig. 6.1: Service hierarchy

turn the four atomic services on and off at once. As far as the “PCS functionality” is concerned, we analogously decompose it into a service which is responsible to turn it on and off (“PCS on off”) and the remaining functionality. The latter can be distinguished into two services which take place if the collision time is smaller or not small than a given limit. With help of the “ACC on off suspended” (“PCS on off”) service, the “ACC core behavior” (“PCS core behavior”) can be turned on and off, therefore we introduce an “enable” and a “disable” relationship. The “Follow Up Control” and the “Constant Speed Control” exclude each other. This is represented by the “XOR” relationship. Analogously, we have an “XOR” relationship between the sub services of the “PCS core behavior”. If the collision time is smaller than a given limit, the ACC functionalities are suspended. This is indicated by the “suspend” relationship. Furthermore, we have two data dependencies in our service graph. The distance mode (“distance-to-be”) is determined by the “Change Follow Distance” service and read by the “Follow Up Control” service. The target speed (“target-speed”) is set by the service “Change Target Speed” and read by the “Constant Speed Control” service.

Formal specification of atomic services The identified atomic services are now formalized by means of AutoFOCUS Mode Diagrams. As services are partial pieces of behavior, one might not want to specify the output on each channel. Therefore, if no output is specified, no restriction is made for this output channel. The “ACC on off suspended” functionality is given in Figure 6.2. Basically, it is comprised of three states (modes) and the transitions between the modes as specified in the requirements. The “Change Follow Distance” service is given in Figure 6.3. If the driver presses the distance mode button, the modes are entered sequentially. In each mode, the respective distance (which the car is supposed to have to the detected vehicle) is sent

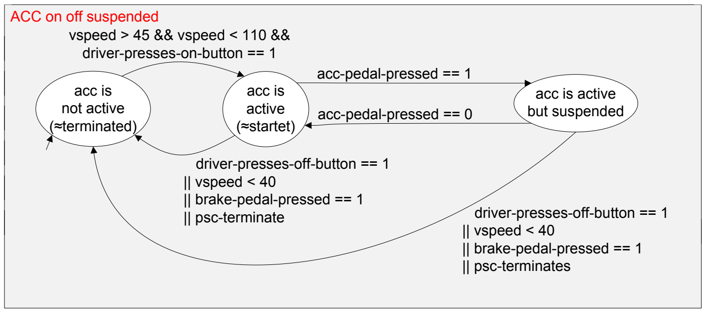

ACC on off suspended
vspeed > 45 && vspeed < 110 && driver-presses-on-button == 1
acc is not active (≈terminated)
acc is active (≈startet)
acc is active but suspended
acc-pedal-pressed == 1
acc-pedal-pressed == 0
driver-presses-off-button == 1 || vspeed < 40 || brake-pedal-pressed == 1 || psc-terminate
driver-presses-off-button == 1 || vspeed < 40 || brake-pedal-pressed == 1 || psc-terminates

Fig. 6.2: Formal specification of "ACC on off suspended'

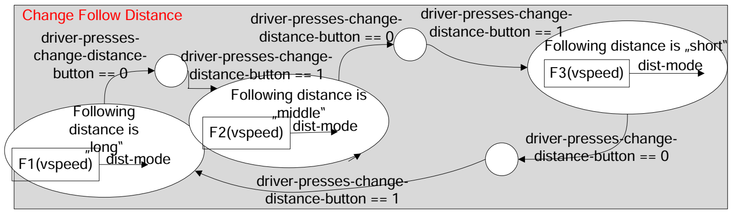

Change Follow Distance
driver-presses-change-distance-button == 0
driver-presses-change-distance-button == 1
Following distance is „short"
F3(vspeed)
dist-mode
Following distance is „middle"
F2(vspeed)
Following distance is „long"
F1(vspeed)

Fig. 6.3: Formal specification of "Change Follow Distance'

to the "Follow Up Control".

Combination of services on basis of the service relationships So far we have formally modeled the atomic services of the case study. Now, we combine the model on basis of the service relationship. The "ACC core behavior" is only executed if the ACC is active. Therefore, the behavior is assigned to the mode "acc is active". Furthermore, the "ACC core behavior" is comprised of the formal specifications of the four sub services "Follow Up Control", "Change Follow Distance", "Change Target Speed", and "Constant Speed Control". The mode automaton assures that the "Follow Up Control" and the "Constant Speed Control" mutually exclude each other (depending on if a target vehicle is detected or not). The other two services ("Change Follow Distance" and "Change Target Speed") are combined in parallel. As far as the PCS is concerned, its core behavior ("PCS core behavior") is only performed in the mode "pcs is active". The XOR relationship between the services "Collision Time Smaller Than Limit" and "Collision Time Not Smaller Than Limit" is realized by a mode automaton. By sending the internal action "psc-terminate" to the ACC (to be more precise to the "ACC on off suspend" service), the ACC is terminated if the PCS is executing the brake.

7 Modeling the Usage Environment of Interactive Systems

Wassiou Sitou
sitou@in.tum.de

7.1 Motivation

While constructing an interactive system, a central issue is to deal with complex interaction patterns rather than complex computations (as in the case of more control-oriented functions): the reactions of the system under construction can vary essentially on the stimuli provided by the environment. Therefore, a thorough exploration of possible combinations of environmental inputs is fundamental. Such an exploration of the environment in which the system will be used, including the structure and the behavior of the contained actors, is referred to as environment modeling. It helps in defining the usage situations of the system, and thus characterizing the possible inputs to the system.

The main objective of environment modeling consists in exploring the usage environment of the system under construction by:**1. Building up a comprehensive model of the intended usage environment:** This model may include environmental entities, their internal behavior and their interrelations in the usage environment. To build up such a model, one should consider the relevant facts regarding user characteristics and actions, environment conditions and interdependencies between them, according to the usage scenarios of the system. With system user we refer to human beings as well as further technical systems (the so called adjacent systems); and **2. Defining the system's boundaries** and thus what are the possible communication points between the system under construction and its environment. This implies the modeling of users characteristics and usage behavior, their performed actions while interacting with the system and also the environment conditions under which the actions are performed. Environment modeling, as part of an overall model-based approach to software development, aims at exploring:

1. the environment actor in high level use cases and thus investigating the origin of the inputs to the system and identifying further use cases that were so far not considered.

2. reasonable flows of actions, which may lead to system inputs, and thus concentrating on such flows to (1) restrain testing parameters to the essentials and (2) take the useful, wished and modeled usage situations into account.

In the following we describe the RE-CAWAR [Sitou and Spanfelner, 2007, Sitou, 2009, Sitou et al., 2009] approach to environment modeling and apply it to the smart key entry system.

7.2 The RE-CAWAR Approach to Environment Modeling

The essential of environment modeling consists in identifying the interrelated conditions in which the system under construction will be used. This conditions are refered to as *context*. Context modeling for interactive systems (a.k.a. environment modeling) means identifying and characterizing the different usage situations of the system and mapping them onto an overall context model. Thus, each instance of the context model represent a usage situation (more precisely, an equivalence class of usage situations) of the system. In each usage situation, a couple of inputs are provided to the system, which may consequently produce some outputs as reaction to the inputs. Such a usage situation is characterized by a set of elementary statements, called *facts*[Falkenberg, 1976]. They reflect interdependencies between the different constituents of the environment. To characterize usage situations, we consider facts of the following schema: "*users* perform *tasks* under specific *environmental conditions*". The performance of a task may require specific actions. While performing these tasks, the system under construction can be used, and this may lead to the occurrence of *interactions* between the system and the user. Thereby, a couple of objects are manipulated. These objects are either part of the environment or part of the system. Those objects that are part of the system are used for the *representation* of the interaction between the user and the system. A well known technique to model facts at a conceptual level, such as in requirements engineering, is the *Object-Role Modeling (ORM)*[1] [Halpin, 2001].

7.2.1 Conceptual Environment Modeling Schema

To systematize the development of an environment model, we have elaborated an iterative modeling schema *(the Conceptual Environment Modeling Schema – CEMS)*. A prerequisite of CEMS is the use of ORM or a derivate as description technique. Environment modeling as such, should be conducted as part of a thorough methodology to analyse the requirements together with the usage situations. Hence, the environment modeling task already requires a good portion of the principal use cases to be identified.

Step 1: Facts identification and transformation: This step consists of identifying interdependencies between the different dimensions of a usage situation (users, their tasks or activities, objects and conditions in the operational environment) by means of use cases, and formulating elementary facts on their basis. The identification of the interdependencies requires a thorough understanding of the application domain. Domain experts should check the quality of the identified use cases. Concrete uses cases (so called usage scenarios with concrete data records) should be identified, whereby their ordering is irrelevant. The identification is carried out along with the transformation of the use cases into atomic facts. Thereby, facts types are to be defined. The defined fact types reflect the overall structure of the considered facts. Atomic facts are of the form "*an object o has a property p*" or "*several objects $o_1 \dots o_n$ are involved in a relationship r*", whereby the relationship can not be expressed in terms of a conjunction of

[1]The Object-Role Modeling (ORM) is a modeling technique similar to Entity-Relationship, with the advantage of abstracting from design decisions while modeling the circumstances of a given case. ORM is tool-supported and is e.g. incorporated in Microsoft Visio.

further facts. A statement like *"Mr. Sitou carries his smart key and approaches his car"* represents a conjunction of two atomic facts, namely *"Mr. Sitou carries his smart key"* and *"Mr. Sitou approaches his car"*. As a guideline, the examined interdependencies have the following form, which represent the types of the facts (fact types).
1. The user u performs the task t;
2. The performance of the task t occurs under the conditions (ex, ey, ez) of the environment;
3. The performance of the task t requires the interaction i;
4. The interaction i takes place between the user u and the system or another users $u\acute{}$;
5. During the interaction i the object o as part of the environment or system is manipulated;

Step 2: Model representation and validation: Within a next step, the identified fact types are to be graphically represented and validated with further samples of data-records (so called data use cases). One or more facts represent a usage situation. The identified fact types, including the object types, predicates and roles constitute the overall environment model. This model is represented at a conceptual level in terms of the ORM-notation (see [Halpin, 2001] for further details). The data use cases are required for subsequent validation tasks. The model is systematically analyzed during the validation. The main attention is given to the predicate in each fact. For each predicate at least one data use case is provided. We should note that the data use cases serve only to validate the fact types and are not part of the model.

Step 3: Model optimization: In principle, facts could be combined. Consequently, fact types could be combined, too. Thus, a fact type could have an arbitrary length, i.e. it could contain as many roles as needed. Anyhow, we recommend to avoid fact types with more than four roles. The predicate of such a fact type should be transformed into smaller predicates containing less roles. This is done during the optimization of the model. Thereby, fact types are combined or – more frequently – broken apart. Accordingly, fact types are added to or removed from the context model, respectively.

7.2.2 Enhancing the Modeling Schema for Technical Issues

The CEMS, as presented above, has been enhanced by a further step to allow for model simulation and help for instance in model-based testing. The principal motivation behind the enhancement is the use of adequate simulation tool to explore the modeled environment. Such a simulation is necessary to validate the soundness of the model. This should be done together with the application owner, who may definitively have a well understanding of the systems intended usage situations, goals and requirements.

Step 4: Implementation in a simulation environment: This step consists in implementing the fact model into a simulation environment, where the overall usage environment is considered as a system. The implementation consists of structuring the environment as a system (component network), modeling the behavior of the involved component, and simulating the overall result.

Step 4-a: Structure the environment: This is carried out like the structuring of a software system. The objects types contained in the fact model are potential component in the structure diagram of the environment. Predicates represent communication channels between the components. The roles played by each object type, which is involved in the object/role pair, represent the communication ports of the concerned component. The result of this step is a network of components representing the structure of the environment.
Step 4-b: Specify the behavior of each environmental components: This is a crucial task. It requires a deep understanding of the application, and thus the involvement of the application owner is of a high importance. The users of the system under construction, the used objects or entities to interact with the system, and further interaction partners of the system are commonly the environment components. These components should be specified at this stage. The specification of technical systems, such as motor engine, with regard to its interaction with the system under construction, is more or less straightforward. This statement holds also, at least to some extents, in case of the used objects and entities to interact with the system. But in case of the users (in the sense of human beings), it is not an obvious task. It is well known that human beings generally behave in a non deterministic way. This should be considered while specifying their behavior. This could imply that different models of the systems users are developed, according to their experience and usage behavior as done in the field of user modeling.
Step 4-c: Simulating resulted model: The structuring of the environment followed by the specification of the included components results in a simulatable environment model. For validation issues, the developed environment model is simulated and possible enhancement are conducted. To simulate the model, a set of IO values pairs are defined. This set may contain not only expected values but also not-expected values, to make sure that the model reflects the reality.

7.3 Application to the SmartKeyEntry Case Study

The proposed methodology has been applied to the SmartKeyEntry system [Feilkas et al., 2009b]. In the following we briefly present the results.
Step 1: Facts identification and transformation: Some identified facts to characterize the usage situations of the smart key entry system are:
1. Mr. Sitou shuts the Driver Door.
2. Mr. Sitou carries the Smart Key 231.
3. Mr. Pfaller touches the Passenger Door Handle.
4. The Smart Key 231 is present in the External Senzing Zone.
5. Mr. Sitou touches the Trunk Door Handle.

The transformation of the above facts (characterizing usage situations) into fact types delivers:
1. **User (user_ID)** shuts **Door (door_ID)**.
2. **User (user_ID)** carries **Smart Key (SKey_ID)**.
3. **User (user_ID)** touches **Door Handle (Handle_ID)**.
4. **Smart Key (SKey_ID)** is present in **Sensing Zone (Zone_ID)**.
5. **User (user_ID)** is of type **User Type**.
6. **User (user_ID)** presses **Wireless Key (WKey_ID)**.

Step 2: Model representation and validation: The conceptual model of the environment, as shown in fig. 7.1 represents an excerpt from the usage environment of the system represented in ORM using the Microsoft Visio CASE tool. Central elements of these model are the user, the smart key, the wireless key, the sensing zone and the door handle as part of the car. The user is the main actor. She could be a DRIVER or PASSENGER. These are the possible values of a user type. A user can carry a smart key. To trigger systems reactions (e.g. Door Unlocking Scenarios), a user can (1) press the wireless key identified by a unique ID, or (2) touch a door handle identified by a unique ID. Alternatively, the systems reactions could be triggered if the smart key is present in a sensing zone. Sensing zones could be EXTERNAL, CABIN or TRUNK. Each Sensing zone has a predefined range [SPECIFY RANGE-VALUE]. Door Handles could be a Driver Door Handle, a Passenger Door Handle or a Trunk Door Handle.

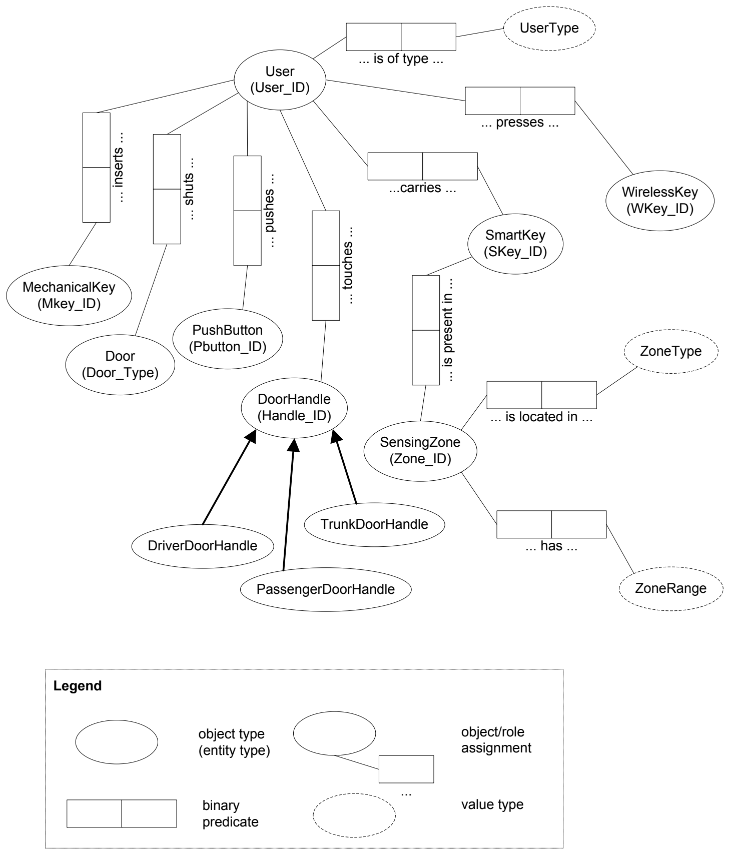


Fig. 7.1: Application: conceptual model of the usage environment

Step 3: Model optimization: Initially, we have considered only the driver as unique user of the smart entry key system. During the validation we have realized, that there are some action of the passenger that may influence the smart key entry system's behavior. As a consequence of the remarks provided by the application owner, we model the passenger and the driver separately.
Step 4: Implementation in a simulation environment: Environment modeling has its use not

only in requirements elicitation and analysis, but also at later development stages. For instance, it could be used to reduce the overall set of test cases. For this issue, a more technical view on environment modeling may be required. Therefore, we model the environment as a system in AutoFOCUS 3, using components that may communicate over port. The behavior of each component is specified by means of state machines. As modeling tool, we use the AutoFocus3 CASE tool. Fig. 7.2 shows the component network that results from structuring the environment. The behavior is specified by means of state transition diagrams.

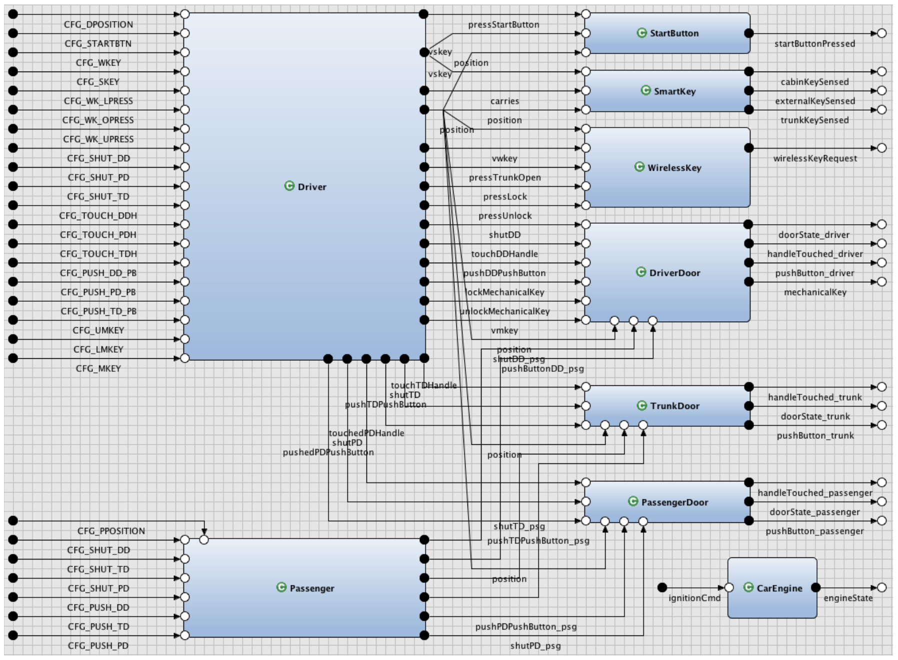


Fig. 7.2: Application: usage environment as network of components

7.4 Conclusion and Outlook

From the methodological point of view, an explicit environment model helps to both *improve the quality* of the system under development and to *increase the efficiency* of the development. The systematic and thorough exploration of possible usage scenarios helps to point the requirements engineer to 'corner cases' of unexpected user behavior or platform failures, ensuring a better usability and stability of the system. Furthermore, it helps to focus the development process to relevant system behavior, e.g., ruling out irrelevant functionality of the system.

While not explored at the current stage, environment modeling can be used in combination with formal verification, e.g., to ensure that no unexpected but possible user interactions have been overlooked, or to ensure safety of usage of the system, e.g., to avoid mode confusion problems as in avionic system. Furthermore, by extending the environment model with typical usage profile data, it can, e.g., also be used to automate the stress test of a system.

8 Model-Based Analysis of User Interfaces

Sebastian Winter
winterse@in.tum.de

8.1 Motivation

Electronic devices are constantly becoming more powerful and therefore more capable of providing a variety of different features. At the same time their size is being reduced, also constantly, by the manufacturers. This development has a considerable impact on the user interface (UI) of such devices. The high number of functions that have to be controlled, the limited size of the display, as well as the limited set of possible haptic commands require new user-interface concepts. Today's interactive devices are therefore complex: They are multimodal and multifunctional; they filter the displayed information and assign actions to the user's commands according to the situation.

In particular, the quality attribute "usability" plays a significant role.

Some defects, however, that limit the usability of interactive devices are not found until the late stages of the software development process, e.g. implementation, integration, or maintenance. Significantly, the later defects are identified, the higher the costs for their correction [Boehm, 1981]. One of the following mistakes is usually the reason why usability defects are not found: Either the nonconformity of specified usability requirements was not detected or some usability requirements were not specified – the so-called implied needs. In the following, we discuss several causes that lead to these mistakes:

- *Some feature interactions cannot be observed if prototypes are incomplete.* Often, prototypes only implement certain parts of the overall functionality. If such a prototype is the subject of an evaluation, requirements regarding certain feature interactions can neither be identified nor verified. Therefore, in order to analyse feature interactions, a multifunctional prototype is needed.

- *Poor diversity of evaluation methods.* The more a variety of evaluation methods is applied, the more likely defects are discovered [Wagner, 2006]. If the focus lies on one evaluation method only, it is likely that certain defects will not be discovered. User tests, for example, are indispensable for the evaluation of interactive devices; however, they exhibit the shortcoming that the inconsistent behaviour of user interfaces (UIs) is often overlooked [Jeffries et al., 1991].

- *Informal description techniques.* The commonly applied description techniques (task models, scenarios, etc.) are informal or semi-formal.[1] Requirements that are specified with an informal description technique are often ambiguous and difficult to check for inconsistency. Moreover, informal requirements allow the developer a considerable freedom of interpretation. This is in particular a problem for the assurance of usability, since the average developer does not have comprehensive knowledge in this field.

The main goal of this approach, as proposed in [Winter, 20XX], is to improve the quality-assurance methods in the early stages of the software development process. Therefore, a model-based method for the analysis of UI designs is presented. The approach is centered around a formal model of the human-maschine system, i.e. the on-going interaction between the user and the interactive device is described in a structured and precise manner.

Since it is possible to create such an abstract model early in the development of interactive devices, it is also possible to conduct an effective analysis of the usability requirements. By doing so, software development costs can be reduced, particularly, due to the fact that this method is based on an automatic verification technique, reiteration of the assessment is possible. Moreover, building formal analysis models improves the objectiveness of the evaluation. Objectiveness is necessary, since the development of interactive devices includes trade-offs between requirements regarding usability and requirements regarding other quality attributes. Additionally, the application of formal description techniques provides a basis through which the conceptual models of the various domains can be integrated [Dix, 1991, Harrison and Timbleby, 1990].

Precise models not only aid in understanding the usability requirements, but also can be re-used in later stages of the development process, as, for example, to derive constructive test cases [Benz, 2007]. In order to provide effective tool support, a mature model theory is a *sine qua non* [Broy and Rumpe, 2007]. For a better integration of usability engineering techniques into the software development process, a seamless modeling approach is needed.

8.2 A Model-Based Approach for the Analysis of User Interfaces

This approach is based on an automaton-based description of the system [Broy et al., 2007]. Driven by a global pulse, a set of automata carry out actions synchronously. However, an automaton can not react to an action until the next step takes place. Thus, the fact that the execution of an action takes time is taken into consideration.

The communication between automata occurs in an asynchronous manner and without explicitly storing the messages. In each step, an automaton can send messages without blocking itself. The receiver, in the following step, can either use an incoming message or neglect it. Since the messages are not stored explicitly, it is possible to apply analysis techniques that require a finite state space, as, for example, model checking. In addition, with respect to model checking, this model of execution has the benefit that the state space is not enlarged because of nondeterministic scheduling.

[1]Examples for informal task models are *Concurrent Task Trees* (CTT) [Paterno, 1999] or *use-case diagrams* related to the *Unified Modeling Language* (UML) [Booch et al., 1999].

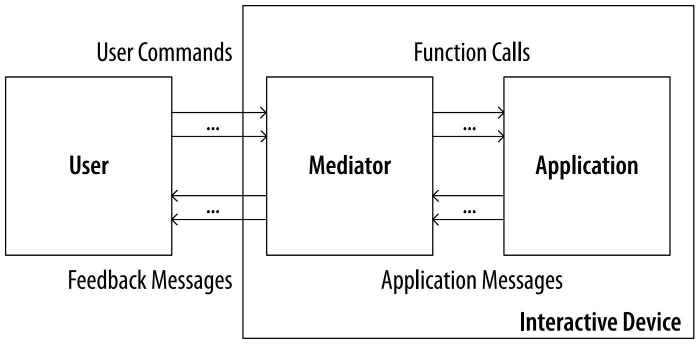


Fig. 8.1: Components of the overall system (based on [Green, 1985])

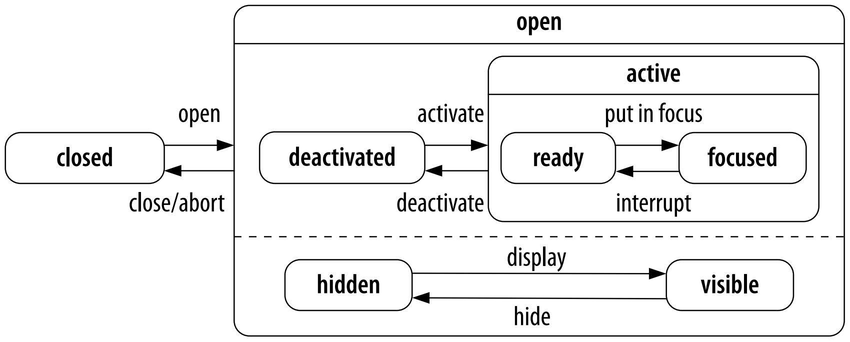


Fig. 8.2: States of a dialog

Taking this fundamental concepts as a base, we define a system model that integrates further concepts from [Henzinger, 2000, Schätz, 2006, Schätz, 2007]. A mapping to the input language of the model checker NuSMV [Cimatti et al., 2002] is defined for this system model. Thus, we have the possibility to verify automatically the system model against a given property.

Based on the system model, we define a notation that describes the automata as hierarchical state-transition diagrams. This notation relates to *Statecharts* [Harel, 1987] and to mode diagrams as in [Alur and Grosu, 2004].

8.2.1 A Conceptual Model of Human-Computer Interaction

This approach combines the central concepts that describe menu-driven UIs within a conceptual model. The overall system is divided into the parts “User”, “Mediator”, and “Application” (see Figure 8.1). The interaction between the user and the interactive device is structured by means of *dialogs*, *dialogs threads*, and *dialog steps*. We define the states and the transitions of a dialog by a life-cycle as depicted in Figure 8.2 and consider a menu as a specific dialog. User commands are divided into explicit and implicit user commands. Implicit user commands can be used to model assumptions of user intentions. This aspect is described in detail in [Broy et al., 2009].

The effect of a user command is categorized by means of the conceptual model. To determine the effect of a user command, the triggered function calls are also taken into consideration.[2] Hence, the *modi* of a user interface design regarding a user command can be clearly defined. A *modi configuration* contains a set of modi each related to a user command and identified by a number.

A dialog step starts with a user command, which is followed by actions of the mediator and the application. These actions are hidden from the user, i.e. he cannot observe them. A dialog step ends with a feedback message from the mediator to the user. The feedback message indicates the current modi configuration of the interactive device.

Typically, most user commands can be considered as a response to the mediator's feedback message. Then, the feedback message constitutes a so-called prompt.[3]

The Standard ISO 9241-110 presents seven "dialogue principles": suitability for the task, self-descriptiveness, controllability, conformity with user expectations, error tolerance, suitability for individualisation, and suitability for learning.

For example, the principle "self-descriptiveness" is described as follows:

> *At any time it should be clear from the interaction in what dialog and in what step the user is located and what user commands are possible.*

In order to achieve the objective of assuring the usability of a UI design, it is necessary to determine those properties that have a postive impact on the usability. Therefore, we derive test criteria from these seven principles. For example, a test criterium for the principle "self-descriptiveness" is:

> *The effect of a user command as defined in the present modi configuration should be inferable from the display. That means, the display should indicate the modi configuration.*

8.2.2 Building Analysis Models

We extend the hierarchical state-transition diagrams in a manner that allows for the modeling of the identified UI concepts. The organisation of the composite states reflects the dialog structure. For example, the activation of a certain composite state corresponds to the activation of a dialog. Hence, a technique is provided that allows for the structured modeling of UIs. Here, the main focus lies on the structured description of the interaction behaviour and not on the graphical design of the user interface. For this model properties were identified and formalized by means of the *Computation Tree Logic* (CTL) [Clarke and Emerson, 1982].

8.3 Application to the "E60" Case Study

To demonstrate the practicability of the approach, we carried out a comprehensive case study from the automotive domain. The case study shows the development of the models and their

[2] In general it is important to analyse the relation between the structure of the dialogs and the functionality of the application.

[3] Developers do not pay as much attention to this reflection for their current UI designs as to command line-based dialogs.

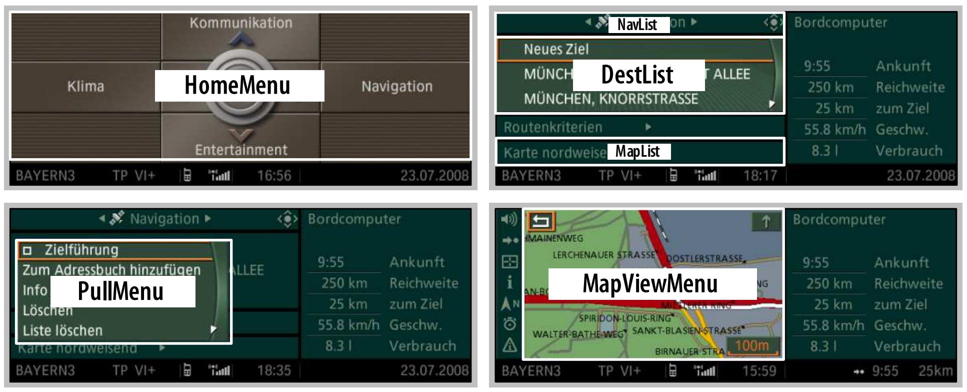


Fig. 8.3: Screens from the “E60” case study

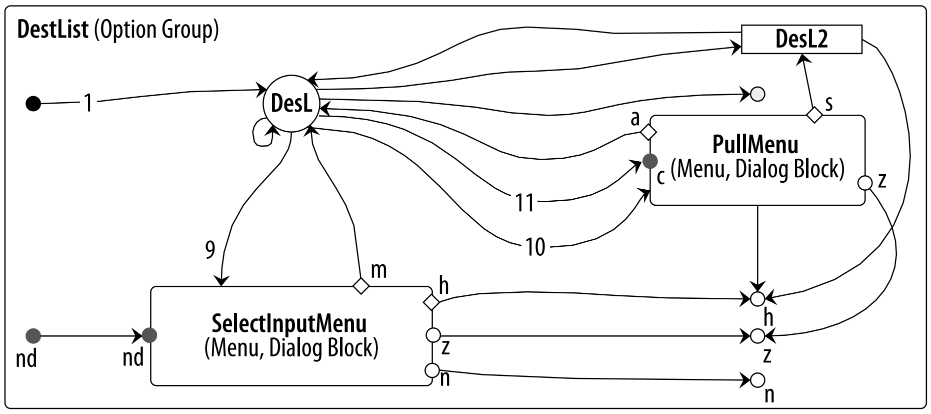

DestList (Option Group)
DesL2
DesL
1
a
s
PullMenu (Menu, Dialog Block)
c
z
11
10
9
m
h
SelectInputMenu (Menu, Dialog Block)
nd
nd
z
n
h
z
n

Fig. 8.4: Diagramm für “DestList”

analysis. The subject of the study is the iDrive System “E60” that is developed by BMW. This system was incorporated into the vehicles of the 7 Series in 2001 for the first time. Altogether, the iDrive System controls 700 functions, which are subdivided into the groups navigation, communication, climate control, and settings.

Figure 8.3 shows screens from the selected part of the design. The “HomeMenu” screen is displayed within the “Home” dialog thread. The remaining screens are displayed within the “Navigation” dialog thread.

First, the “Home” dialog thread is put in focus. Accordingly, the start menu (“HomeMenu”) is shown to the driver. By moving the control knob to the right, the navigation menu is called. Here, the list “NavList” that contains the functions regarding navigation is shown to the user. By pressing or moving the control knob downwards, the “DestMenu” submenu is called. This menu is divided into two *option groups* “DestList” and “MapList”. As a default, the list of the driver’s destinations (“DestList”) is displayed. If the route guidance is active, the chosen destination is marked by an arrow. If the driver activates a destination option, the pull-down menu “PullMenu”, a submenu of “DestList”, is called. The driver can activate or deactivate the route guidance by means of “PullMenu”. If the driver does not carry out a user command within a certain time span, the pull-down menu is aborted. As a example, Figure 8.4 shows the model of the “DestList” option group.

The case study shows that the approach is capable of identifying the defects of a UI design with respect to the test criteria. The results support the experts’ criticism on the iDrive System’s design.

8.4 Conclusion and Outlook

In this chapter a method of analysis that describes human-machine systems by means of discrete mathematical models was presented. The key concepts for the description of menu-driven user interfaces were combined within a conceptual model and integrated into an automaton-based description technique. Properties of the analysis models that influence the quality attribute “usability” are formalized by means of temporal logic.

The main purpose of this approach is the structured description of the interaction behavior, rather than the graphical design of the user interface. Essential to this end is the domain-specific structure of the model. Therefore, we concentrate on menu-driven user interfaces for electronic devices, e.g., automotive navigation systems or mobile phones. As an input, an informal description of a human-machine system or a prototype of a user interface has to be provided.

This approach supports the analysis of usability requirements through structured modeling of human-machine systems and simulation. In addition, with the aid of model checking the conformity of the models to these requirements is automatically verified. This provides an unbiased means for the evaluation of user interfaces.

9 DeSyRe – Decomposition of Systems for Reuse: Transition from System to Subsystem Using a Criteria Catalogue and an Artefact Model

Birgit Penzenstadler
birgit.penzenstadler@in.tum.de

9.1 Motivation

The current state of practice in automotive software development at an original equipment manufacturer's site is to first produce a system specification and then to separately produce requirements specifications for the subsystems that are to be assigned to the suppliers for distributed development. This course of action is time-consuming and costly as systematic reuse is not yet applied within these two process stages. There is a number of challenges that pose demands for DeSyRe approach: The challenge for appropriate requirements documentation, system decomposition, architecture documentation, and system integration [Broy, 2006].

The research questions with respect to decomposition are: What are the influencing criteria on system decomposition? How is a subsystem distributed within the system across the different abstraction layers? How is a subsystem described across the abstraction layers?

The part about system specification is concerned with the questions: How is a system decomposed for the suppliers? How can the requirements from a higher abstraction layer be transitioned into the requirements for the next lower abstraction layer? How are these requirements documented? What dependencies result from the requirements for and in between different subsystems? What types of constraints arise for the subsystems? How do these constraints influence system integration?

9.2 The DeSyRe Approach: Decomposing a System and Refining the Requirements

The contribution of this work is to provide an answer to how one overall system specification can be used to extract subsystem specifications. This is achieved by an **artefact model** and a **guiding process** from decomposition to subsystem extraction.

A criteria catalogue for the decomposition of systems into subsystems lists possible influences that have to be taken into account during the decomposition process. A supporting artefact model provides the framework for representing and documenting the context, the requirements, and the specification of the system and its subsystems. A process guides the usage of the catalogue and the artefact model. A case study shows the application of the approach.

9.2.1 Decomposition Criteria

The decomposition of a system is the first step into direction of defining the architecture after the analysis of the requirements. For actually performing such a decomposition, it has to be shown *how* to decompose a system, *what criteria* to consider and how to apply them.

The decomposition process is influenced by system type-specific aspects as well as domain specific and individual constraints. The reference framework reflects both points of view in four criteria categories that guide through the decomposition process. Apart from aspects depending on the type of system and the domain specific requirements, there are four categories of criteria. All four categories have to be considered during the process of requirements engineering. The end user's point of view provides the *functional criteria* and the *quality criteria*. The *technical criteria* derive from the constraints given by the technical solution domain and the future system environment as well as from architecture design rules. The stakeholders of the *directive criteria* will vary depending on the business view — apart from laws and standards there can be licensing constraints, business rules and influences from business-supplier-relationships.

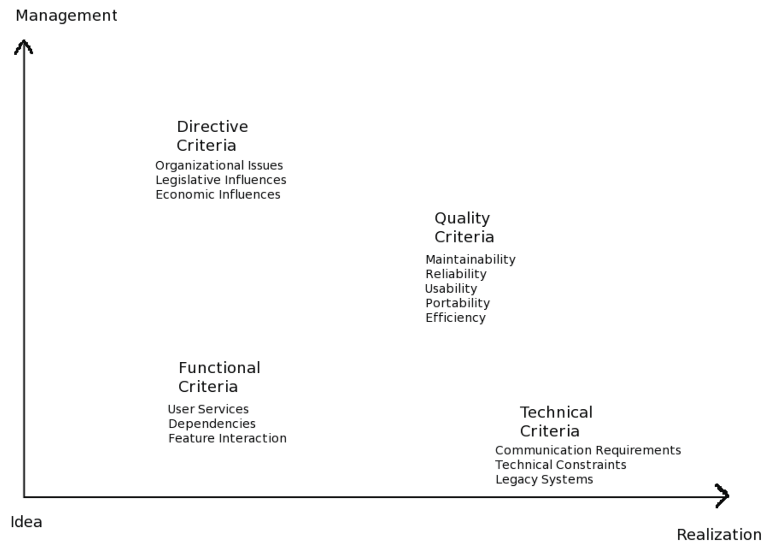


Fig. 9.1: Taxonomy and overview of the criteria catalogue

A taxonomy and overview of the criteria catalogue is given in Fig. 9.1. The criteria were found in information sources from requirements engineering activities and related reference models (e.g. [Geisberger et al., 2006]), via the interviews of a small conducted field study, through many discussions with colleagues, and further literature research, inter alia Conway [Conway, 1968] and Herbsleb [Herbsleb and Grinter, 1999], Nuseibeh [Nuseibeh, 2001], Parnas [Parnas, 1972], and Wojcik [Wojcik et al., 2006]. Highly theoretic approaches on formal decomposition, as performed by Abadí and Lamport[1], are not further considered.

Due to the aforementioned dependency of the criteria on the type of system and its specific situation, there is no general rule on how to incorporate all the criteria equally into the decision for the division into subsystems. The relevance of the criteria has to be evaluated separately for each system and according to that relevance, the division can be decided individually. It is not desired to limit the general applicability of the approach but, to be able to justify the selection of criteria for the division, it is necessary to assume a certain type of system in the following. The catalogue of criteria will be the same but the criteria's priorities might vary for specific systems or types of systems.

The type of the specified system determines to what extent each of them is relevant for its division into subsystems. In the work at hand, it was already defined that the system type of the given context is embedded systems.

9.2.2 Artefact Model

The decomposition criteria give an idea of how to analyze a system and draw a first coarse-grained sketch of the architecture. The other half for successful requirements engineering and design consists of modelling and documenting the system and its environment appropriately. The artefact model was developed as an easily applicable reference model for exactly that purpose. It features both a differentiation of three abstraction layers and three content categories, resulting in a 3*3-matrix structure with nine artefact classes, described and depicted in the REMsES chapter (Chap. 2) of the work at hand.

9.2.3 The DeSyRe Guiding Process

DeSyRe is an abbreviation for "**De**composition of **Sy**stems for **Re**use". The process is intended as kind of application guideline that describes the systematic usage of the decomposition criteria analysis and the artefact model. An artefact-centred approach is considered more valuable for strongly distributed software development as present in the automotive domain than prescribing smallest process steps.

A system with widely distributed development includes many development teams and therefore is hardly likely to be developed using the exactly same processes and tools. This leads to the pragmatic solution of defining a concrete artefact model with assigned contents but a less strict process for how to develop the artefacts.

The method DeSyRe describes the decomposition into and seperate consideration of subsystems for the purpose of distributed development and subsequent integration as well as reuse of

[1]Abadí and Lamport analyze the formal decomposition of a system in their paper "Decomposing Specifications of Concurrent Systems" [Abadi and Lamport, 1994] on which they base a formal conjunction of component specifications for program verification.

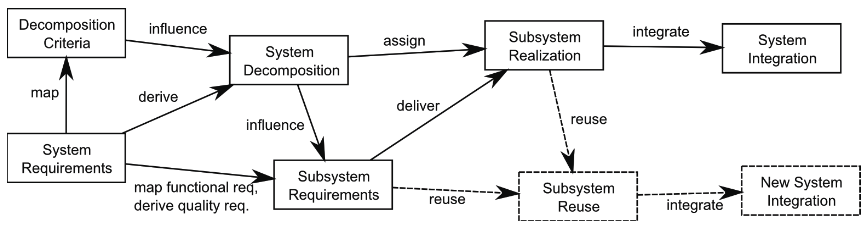


Fig. 9.2: Process DeSyRe

subsystems as depicted in Fig. 9.2.

The process includes the following steps:

1. Decomposition into subsystems: Consideration of the Reference Criteria Catalogue, Dependency Analysis, and Decomposition Realization

2. Transition to subsystem requirements: Context Artefacts, Requirements Artefacts, and Design Artefacts

3. Extraction of desired subsystem: Define borders, Document interaction, and Gather artefacts

4. Integration or reuse of a subsystem: Identify appropriate subsystem(s), Check compatibility, Analyse and integrate additional requirements, and Integrate artefacts

The supported concepts of systematic guidance, checklists, structured artefacts and architecture documentation promise a high effectiveness of the DeSyRe method.

9.3 Case Studies

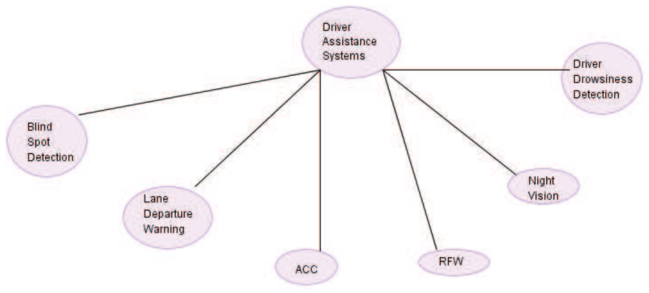


Fig. 9.3: Service Graph Driver Assistance Systems

The driver assistance systems case study, see Fig. 9.3 is a real-life example from the industry. On the system layer, driver assistance systems are considered and then two example subsystems were further detailed to evaluate the approach, namely the active cruise control and the radio frequency warning system.

The “active cruise control” (ACC) is a driver assistance system. More precisely, it is an intelligent speed control system that automatically maintains a pre-defined speed taking into account a minimum distance to the car in front.

The “radio frequency warner” (RFW) supports the driver in coping with the information input from the surrounding environment by use of radio frequency signals. The case study illustrates the usage of the catalogue, the artefact model, and the process.

Due to limitations of space, these two subsystems can not be illustrated further within the short description at hand.

9.4 Conclusion and Outlook

The DeSyRe approach is at the moment being extended in direction to the systematic refinement of requirements.

The method DeSyRe may be integrated with some of the other approaches proposed in the work at hand. The artefact model allows for context modelling as described in Chap. 7 and service modelling according to Chap. 6.

If, additionally or initially, there exist natural language requirements, the approaches using patterns and linguistics described in Chap. 4 and Chap. 5 may find application in parallel to easier derive the requirements artefacts to be modelled.

10 Integrating Software Development Risk Management into Requirement Engineering

Shareeful Islam
islam@in.tum.de

10.1 Motivation

Customer requirements are in development projects often not deeply understood what results in expansions of the system scope, in missing business needs or even in a rejection of the final system. Software risk management can effectively contribute to control these problems before they occur to improve the overall project outcomes. But even though the awareness of the importance of risk management, risk management is not always well-applied in practice. A study showed that 75 % of analysed project managers did not follow any detailed risk management approach [Ropponen and Lyytinen, 2000]. The cause of most project failures has little to do with technical issues despite the common tendency among project managers to focus more on these. Several software risk management approaches and standards emphasise the importance of performing risk management activities as early as possible. However, there is still a lack of comprehensive detailed guidelines on how to integrate risk management activities into a specific phase of the development life cycle. We believe that the integration of risk management into requirement engineering effectively contributes to reduce requirements errors. The reasons is that requirement errors are the most expensive software errors that are numerous and persist through out the life cycle [van Lamsweerde, 2009]. Thus, this work contributes:

- To propose a goal-driven modelling framework to support more effective, systematic and straight forward method and activities for software development risk management from holistic perspective.

- To integrate risk management activities into RE by considering the dependency among the requirement and risk artefacts and inherent similarities among the activities and techniques.

- To develop a goal-risk taxonomy along with questionnaires for effectively identify the risk factors from the early development components.

10.2 Goal-Driven Software Development Risk Management Model

The goal-driven software development risk management model (GSRM) is based on existing goal modeling techniques to accommodate the risk management activities rather than developing new ones. Goal modeling approaches are recognised in the requirement engineering community being useful to elicit, analyse, negotiate, document and modify requirements. GSRM goes beyond the current best practice in GORE by taking a holistic view on risk management. It extends KAOS [van Lamsweerde, 2009] to support risk management activities within RE. The main framework of GSRM consists of four layer modeling structure [Islam, 2009], as introduced in the following.

Goal Layer. Goals are the objectives, constraints and expectations that require attaining within the software development through the cooperation of system agents. GSRM starts with identifying, elaborating and modeling the goals based on the identified development components. This involves several dimensions relating to the elements under the project constraints, process, product, human and internal and external environment. Furthermore, stated factors in the literature used to define the project success are also taken into consideration when identifying goals, as these address stake holder expectations. The goals may initially be high level and then later further elaborate using *and* or *or* refinement into sub-goals. This means that goals can be stated at different levels of abstraction ranging from higher level coarse grained to lower level finer-grained goal expressions. To better aid developers in the earliest stages of a development project where much is uncertain, GSRM provides step-by-step guidelines for goal and sub-goal formulation by following the SMART principle [Mannion and Keepence, 1995].

Risk-Obstacle Layer. Risk-obstacles are the causes that reduce the ability to achieve one or more goals. The layer models these risk factors to specify how they negatively affect the goals. To ease the risk identification in the early requirements phase, GSRM provides a set of general risk factor structured according to goal categories considering the development components, elements and factors. Same risk-obstacle can be relevant to more than one goal and this is important to capture in the risk obstacle layer, as it is crucial information when later considering treatment options. Risk factors that cross-cut several goals are in general more effective to treat, as the treatment effect often then propagates also to goals that are not directly linked to the particular risk factor. In GSRM, we follow a set of questionnaires based on the early development components to identify such cross-cutting risk obstacles.

Assessment Layer. The main role of the assessment layer is to provide an insight into each individual risk obstacle. This includes identifying any resulting risk event from the risk factors. Each risk event is characterized using the two properties: (a) likelihood and (b) impact. What is important to take into account when working on the assessment layer is that the same risk factor may lead to more than one risk event and that the same risk event can obstruct more than one goal. Such representations allow to capture situations where an event is influenced by more than one risk factor and where all these may have negatively impacts on one or several goals. We model these situations using Bayesian belief network by deriving the casual relationship between the risk factors and events [Pearl, 1988, Jensen, 1996]. All risk events and goal relationships are modelled by adding an obstruction link from the risk event to the specific goal it obstructs. In GSRM, the risk analysis explicitly considers how the risk events obstruct the goal as well as posing problems within the development environment. If the risk events are improbable based on the risk metrics value of the associated risk factors then it implies that the confidence of the related goal fulfilment is high. The risk assessment layer finally prioritised the risk based on the likelihood, impact and its influence towards goal dissatisfaction through an obstruction link.

Treatment Layer. The fourth and final layer in the GSRM is the treatment layer, which models the possible control actions and chooses the most suitable ones to mitigate the risk and thereby attain the goals. Once the goals, risk factors and risk events are identified and analysed by the goal risk obstacles and the assessment layers, it is crucial to identify, plan and then quickly implement cost effective counter measures. Therefore, the aim of the treatment layer is to gain control of the risks as early as possible and preferable in the earliest stages of the RE by assigning appropriate counter measures. To visualise the relationship between treatment, risk obstacle and goals, we establish a contribution link from control action to goal. Here, contribution means the ability of the treatment to support the goal by reducing the effect or likelihood of or remove the associated risk factors.

Figure 10.1 summarises the different layers of the framework.

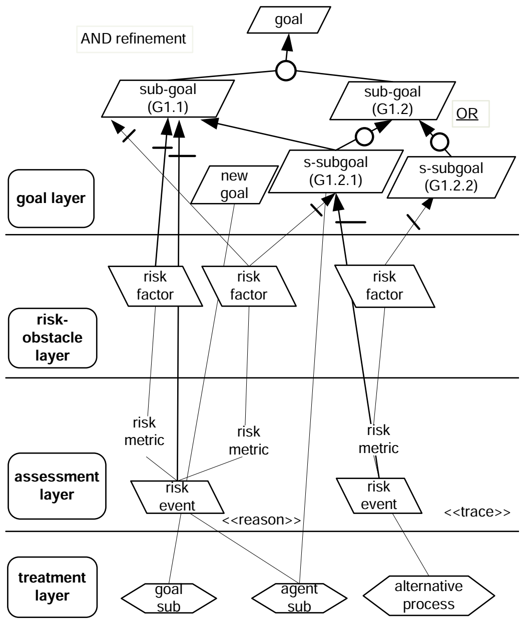


Fig. 10.1: Overview of GSRM

10.2.1 GSRM Activities

Several approaches and standards on software risk management agreed that, to be successful, risk management must be run as a continuous process involving repeated risk assessment and project specific risk mitigation activities throughout the life cycle [Boehm, 1991, Ropponen and Lyytinen, 2000, Kontio, 2001, van Lamsweerde, 2009]. And it is recommended to approve a risk management plan before performing any activities for risk management. We follow this baseline for defining the activities under GSRM. The risk management plan is the initial activity mainly focus to define the scope of risk management. Goal identification and elaboration based on the project system vision, business goals and development components, elements and factors is the next activity after the plan. Risk identification followed by the goal definition includes techniques such as goal-risk taxonomy-based questionnaires and brainstorming sessions to identify the risk factors. The identified risks are cluster into categories according to development component, elements and factors. Risks are then analysed, prioritised through BBN. Risk control actions are then planned and implemented to prevent/reduce the risk. Finally, the risk monitor activity continues by evaluating the effectiveness of the control action and by identifying new risks factors and associate events.

10.2.2 GSRM within the Context of RE

Integration of software development risk management in general and GSRM in particular into RE is one of the main contributions by this research. Therefore, we consider a comprehensive analysis regarding the underlying issues relating to the integration. Our focus is from two different perspectives: *artefact orientation*, *activities along with the roles* require to perform the activities to provide certain rationale within this context [Islam et al., 2009].

Artefact oriented RE is a systematic methodology that describes the problem space of the system to-be as comprehensively as possible and produces several requirement artefacts [van Lamsweerde, 2009]. Requirement artefacts rely on attributes to specify the properties of the artefact and syntax to represent the attribute through textual or graphical based way. Highly structured text is mainly used to represent the requirement artefacts. Risk artefacts also consist of certain predefined highly structured attributes. Graphical visualisation are limited supported by UML use case or activity diagram when representing user requirements. But GSRM requires visual notation to model the goal-risk-treatment, to represent the casual relationship between risk factors and events. Several goals from the project context such as business, stake holder expectations and constraints from the organisational environment are one of the main inputs for eliciting requirement and risk factors. Elicited requirement in particular user and system requirement indeed support to create several risk artefacts such as detailed risk list, goal-risk model, etc. In fact requirements are among one of the elementary inputs for risk identification. The reduction of project risks is a critical requirement for any project. Risk control actions also contribute to goal satisfaction such as an active customer/user involvement, an effective RE process, and finally facilitates the systematic performance of RE activities.

The activities and underlying techniques for both RE and GSRM are interrelated. For instance RE mainly comprised of activities such as elicit, analyse and validate of requirements which further decompose into sub-activities. Risk management activities, as mentioned in Sect. 10.2.1,

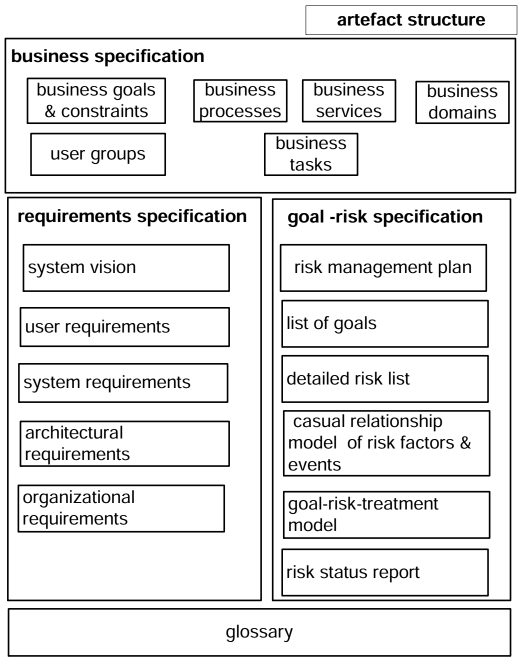


Fig. 10.2: Artefact oriented view to integrate GSRM into RE

also provide similarities. Several techniques within the activities are similar. For instance, requirement, goal and risk identification rely on checklists, brainstorming sessions and several organisational documents. Elicited requirement are reviewed to identify the errors that pose any potential risk. The result of risk management supports to make decision to prioritise requirements. Risk monitor similar like requirement validation is a continuous activity and if require persit throughout the development life cycle. Customer/ user representative in particular members of user groups provide certain input to elicit user requirements, goals and risk factors. We believe both project manager and requirement engineer with certain domain knowledge about risk management can perform the risk management activities under GSRM. Because generally in real project situation, there might not have adequate support to appoint expert for performing risk management activities in particular for project with having limited budget and time constraints. Figure 10.2 outlines the requirement and risk artefacts along with the artefact from the business specification.

10.3 Case Study

The main purpose of the case study is to validate the approach respecting the usefulness of the integration of risk management into RE. To accomplish the task, GSRM is applied to an ongoing software development project. Note that by the term usefulness, we consider the benefits and limitations of performing risk management activities along with requirement engineering activities and dependency of the requirement and risk artefacts within RE.

Study Context. The project context was a development project of an application software with some common features such as a sales system, including inventory, purchase and reporting modules. A bar code reader is the main external hardware interacting with the software for performing sale functions.The project size is 14 man-month with a duration of ten months. In earlier projects at the company, risk management was performed informally without following any detailed process and techniques. Risk factors are considered mainly from a technical and business perspective without any risk analysis and monitor capability.

Instrument. Our initial focus is to identify the goals and risk factors from the early development components. Therefore at first we performed interview sessions based on our goal-risk taxonomy-based questionnaires with some of the development team members. There are about 95 close questions used to interview the participants. Afterwards, a brainstorming session was conducted with the project manager and requirements engineer to revise the response of the interview session along with some project documents. We continued the rest of the activities of the GSRM so that risks are prioritised and certain control actions are implemented to control the risk. Finally, 25 open questions were asked to project manager, requirement engineer, and one developer regarding the usefulness of GSRM into RE.

GSRM activities. A risk management plan was defined with the project manager by mainly reviewing the project scope and the business goals. Note that the company doesn't consider any system vision within the project. Only business goals, business processes and user groups are identified within the business specification. We performed the first interview session when initial user requirements were already elicited. The brainstorming session was very effective to review the risk factors according to several cluster. However risk assessment was done only by us without participation of any member from the development team. This is because they do not have any domain knowledge about the risk assessment techniques and the team was in very tight schedule pressure so that we have not any opportunity to delivery any tutorial about GSRM. But prioritised risks were agreed with project manager and requirement engineering and they participate to determine the treatment plan in order to control the risk events. Finally a risk monitor session meeting was conducted at the end of our involvement to the project after collecting the feedback.

Discussion. There are several observations when performing risk management activities within RE. The participants regarded the activities of GSRM systematic and do not incur any extra burden within RE. Some RE artefacts such as the business specification, goals and requirements closely support risk identification. Risk control actions explicitly reduce the requirements error. We have initially 35 risk factors beign identified that related to more 15 risk events. Among them 40% are related with requirement-related problems. A total of 27% system requirements are erroneous. Some of the requirements are revised to reduce the error by risk control. However not all requirement errors are controlled and we also could not control all risk factors. On the other hand producing risk artefacts such as goal-risk model, detailed of risk factors, risk status report consumed extra time and resource when the project is in tight schedule pressure.

10.4 Conclusion and Outlook

This work proposes a goal-driven modelling framework to manage software development risk since early stage of RE. The model was applied in an active software development project to validate the approach. The result showed that risk management in particular GSRM well-integrated into RE and did not incur any extra burden nor conflict within RE. Therefore, we believe that the integration of risk management into RE supports to assess and manage risk from early development and contribute effectively to reduce requirement errors. However activities like risk assessment, monitoring meeting, producing risk artefacts can increase the estimate allocated schedule within RE when numbers of risk factors are higher. Furthermore risk assessment and continuous monitor are difficult to perform by the development team members when no risk expert in available in the project.

11 Towards an Integrated Requirements Engineering: A Big Picture

Manfred Broy, Leonid Kof, Wassiou Sitou
{broy|kof|sitou}@in.tum.de

11.1 Requirements Engineering in a Big Picture

The overall approach to integrated requirements engineering (RE) is steered by the paradigm of artifact-orientation. According to this paradigm, an integrated RE approach is based on a reference model of different contents that are to be worked out during the RE phase of software development. These contents are organized in so called RE-artifacts. Thus, RE-artifacts are hierarchically structured into content items defining single areas of responsibility and building output of single tasks. In this context, an RE-artifact is a deliverable that is produced, modified, or used by a sequence of tasks in RE.

The above mentioned reference model is inherently abstract and requires domain specific concretizations. Such concretizations result in so called RE artifacts models with mandatory and optional content items. Examples of RE artifacts models are the RE artifacts model for embedded software intensive systems and the RE artifacts model for business information systems. RE artifacts models should be tailorable, i.e. they could be adjusted (some artifacts or content may be left out) to fit current development situation, e.g. according to the targeted organization and project context.

To elaborate and/or make use of RE-artifacts, or even content items, sequences of steps should be performed. These sequences of steps are organized into methods. The RE-philosophy in the TUM I4 RE-group envisages that the methods may be based on a core concept of system modeling with different system views (process, usage, interaction, behavior, data, etc.). An overall approach to RE should cover methods that address central RE-topics such as problem domain analysis, process modeling, formulating requirements, modeling usage behavior, modeling system's usage environment, modeling system's usage interface, analyzing interaction behavior and of course modeling quality requirements. The main challenge in developing an integrated approach to requirements engineering consists of bringing the methods together in an overall framework. Each of the methods requires specific contents in form of input and produces further contents in form of output, which in turn may serve as input to other methods.

In a holistic approach to requirements engineering, we need to address three central issues: *why* a system is needed, *what* needs must be addressed by the system, and *who* will take part in fulfilling such needs. This classification is in the style of the three dimensions of requirements engineering (why, what and who) proposed in [van Lamsweerde, 2009]. To obtain an adequate requirements specification several methods are necessary to address these three issues. For instance we need methods to address:

1. Why a system is needed, e.g. methods to
   - Model the problem domain to identify the goals that are to be achieved by the system
   - Model the identified goals
   - Model processes that are to be supported by the system in order to identify the requirements to the system
2. What needs must be addressed by the system, e.g. methods to
   - Elicit, structure and formulate functional requirements (e.g. using use cases, formulation patterns)
   - Make sure that the formulated functional requirements are complete and consistent
   - Formalize the formulated functional requirements (e.g. using functional specification techniques, service hierarchies)
3. Who will take part in fulfilling such needs, e.g. methods to
   - Model the application domain including its rules and the required data by the system functions (e.g. using data modeling techniques)
   - Model the usage environment and relate usage situations to functional requirements (e.g. using environment modeling approaches)
   - Describe and analyze the interaction of the system with its environment (e.g. using user interface analysis methods)

Furthermore, we need additional methods to address specific issues such as methods to:

- Model quality requirements
- Manage risks that may occur
- Validate and verify the specified requirements
- Decompose the specified systems into reusable subsystems especially in case of subcontraction (e.g. using system decomposition approaches)

These methods are used to address the three dimensions (what, why and who) during the RE phase of system's development, where goals are gradually refined into functional requirements, quality requirements, and constraints. The overall activities, which are carried out during RE, result in the requirements specification document. Based on the requirements specification document a system design is conducted. Especially in case of sub-contraction (implication of customer and contractor, e.g. OEM/Supplier relations), a specification of the subsystem requirements is important. Thus, if necessary, an RE for the subsystem should be conducted. In most cases, RE and system design are iteratively conducted hand in hand. Figure 11.1 shows a big picture of RE as described above.

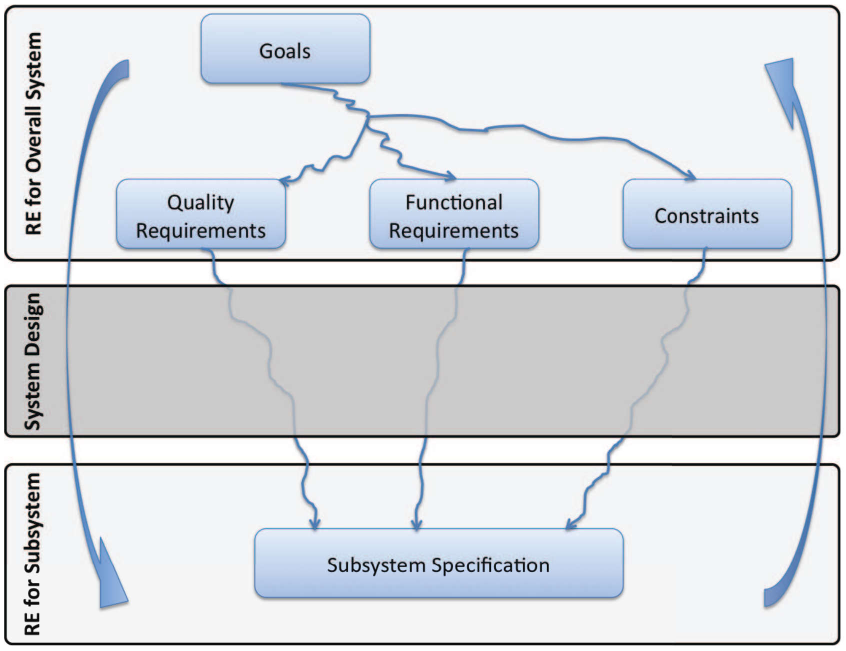


Fig. 11.1: Requirements Engineering in a Big Picture - Simplified

11.2 Integrating Current RE Approaches at TUM I4

Figure 11.2 shows a map of the current situation in RE at the informatics chair for software and systems engineering at the Technische Universität München. This map is organized into three parts: *Reference Contents*, *Artifact Models*, and *Concrete Methods*. Note that we organized the conrete RE methods via pieces of an incomplete puzzle to express overlappings and missing methodical support. The different elements on this map are colored according to their maturity: green means we do cover the topic already, yellow means we are covering the topic currently, and red means that the topic is currently not explicitly addressed in our group, but is on our agenda. Furthermore, the positioning of the concrete methods on the map is not of importance and has no chronological relevance. The naming of the concrete methods comes from the methods' authors and is not necessary related to the above list of methods. The concrete methods on the map are somehow related to each other. The relation depends on what the methods need (input artifacts) and what they produce (output artifacts). Most of the methods require artifacts that are not necessary the same (especially regarding their form) artifacts produced by other methods. Thus, the pieces of puzzle on the map do not fully fit to each other. Also, we should note that we are far away from complete puzzle that may cover the whole RE. An extension of the puzzle remains a challenge for future works.

The RE Artifacts Model for *Embedded Systems (REMsES)* results from the REMsES project. In close cooperation between industrial and academically partners, the recently completed REMsES project has developed a guideline to support RE processes in the automotive industry. The

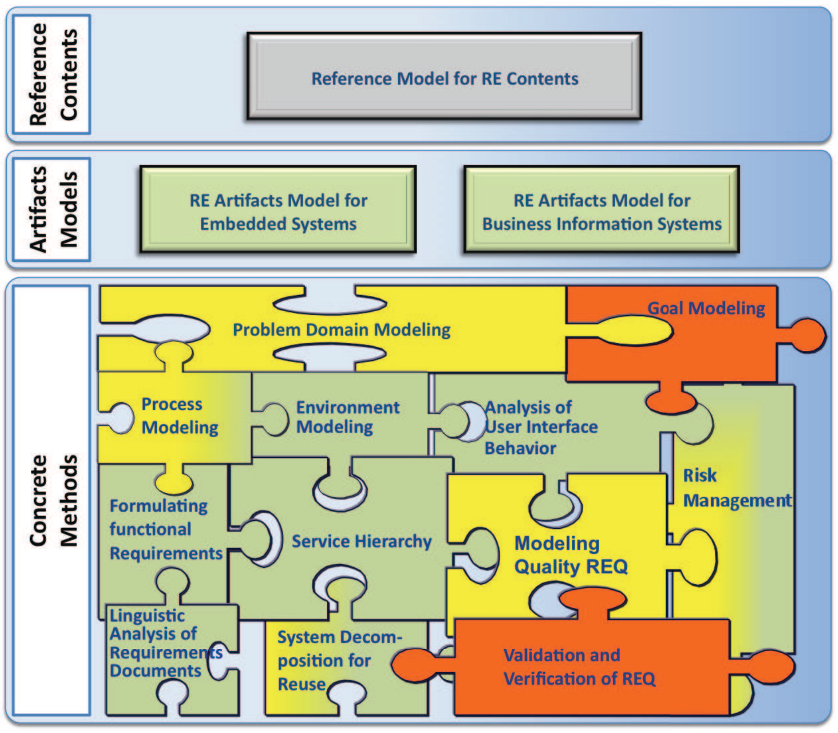

Reference Contents
Reference Model for RE Contents
Artifacts Models
RE Artifacts Model for Embedded Systems
RE Artifacts Model for Business Information Systems
Concrete Methods
Problem Domain Modeling
Goal Modeling
Process Modeling
Environment Modeling
Analysis of User Interface Behavior
Risk Management
Formulating functional Requirements
Service Hierarchy
Modeling Quality REQ
Linguistic Analysis of Requirements Documents
System Decom-position for Reuse
Validation and Verification of REQ

Fig. 11.2: Integrated Requirements Engineering at TUM in Big Picture

guideline enables the requirements engineers to cope with the increasing quantity, size, and complexity of such systems. The chapter dedicated to REMsES in this report presents the major results of the project, namely, the fundamental principles of the approach and the underlying RE artifacts model, the guideline itself, the tool support for REMsES-based processes, and the major results obtained from validation of the REMsES approach.

The RE Artifacts Model for *Business Information Systems (REMBIS)* provides, as the name reflects, a reference approach for business information systems' analysis. It serves as an orientation for producing precise specification documents being conformant to the reference model. Based on this reference model, we define a mechanism for a systematic and transparent customization of the reference approach. The customization mechanism defines how to customize the reference model at organizational environments concerning process-integration. Furthermore it defines how to customize the model at project-levels according to variable project influences.

The concrete method *formulating functional requirement* is about an approach to textual requirements patterns. The approach aims at improving the quality of textual requirements by reformulating them according to textual requirements patterns with a transparent underlying formal model. Hence, the advantages of natural language (e.g. understandability, flexibility, structurability) are combined with the advantages of formal models. The result of the proposed approach is a list of textual requirements that follow a formal model - hence they are quite easy to transform in formats needed by more formal steps of the requirements and design phase.

The concrete method *linguistic analysis of requirements documents* describes computational linguistics approaches to requirements engineering. Since natural language is the main presentation means in industrial requirements documents, typical requirements documents are incomplete and inconsistent. These deficits can be reduced with the aid of computational linguistics. The approaches address two types of deficiencies: terminological inconsistencies and incomplete behavior descriptions. In both cases, deficiencies are detected by extraction of formal models from the text and inspection of the resulting models.

The concrete method *function/service hierarchy* describes an approach to formalizing functional requirements. Based on formulated and structured textual requirements, the requirements are formalized step by step. The core of the proposed approach is the notion of services. Services in this context are system functionalities that are visible to and perceivable by the user. Among services, there exist a couple of relations and interdependencies, which are of importance and should be taken into account while formalizing requirements.

The concrete method *environment modeling* describes an approach to modeling the usage environment of interactive systems. Since interactive systems are fundamentally constructed to interact with surrounding actors (human beings or further technical systems) in their usage environment, a thorough understanding of the usage environment is crucial for designing useful and usable systems. The proposed approach to environment modeling aims at building up a comprehensive model of the intended usage environment and defining the system boundaries.

The concrete method *analysis of user interface behavior* describes human-machine systems by means of discrete mathematical models. The models are notated by a hierarchical automaton-based description technique. The main focus lies on the structured description of the interaction behavior and not on the graphical design of the user interface. The key concepts for the description of menu-driven user interfaces are combined within a conceptual model and integrated into the automaton-based description technique.

The concrete method *system decomposition for reuse DeSyRe* is a self-contained modeling approach aiming at the transition from requirements to design. It describes a subsystem across the three abstraction layers from user functions to technical architecture over logical architecture. An artifacts model for integrated requirements engineering and design gives an appropriate representation and documentation. The guiding method DeSyRe explains the system decomposition with help of a reference criteria catalogue, describes the transition from system requirements to subsystem requirements on the basis of the defined specification artifacts, and enables to extract subsystem specifications from the overall system specification so they can be handled independently (for assignment to suppliers or reuse).

The concrete method *risk management* focuses on the importance of risk management activities at early development phase and proposes goal-driven modeling framework for managing software development risk even in early requirement engineering. The approach models the goals of the system under development, risk factors that obstruct the goals, and the treatment actions that influence the goals. This facilitates the assessment, management and monitoring of software development risk from early requirements engineering on.

The concrete method *modeling quality REQ* is an approach to support the engineering of quality requirements. These are a special kind of non-functional requirements. The approach relies on a quality model that defines a set of properties that influence the various quality attributes. The quality model is integrated with a use-case based approach for eliciting and analyzing quality requirements. Furthermore, the quality model enables the refinement of the quality requirements to eventually get verifiable requirements. Through the link of detailed requirements to quality attributes, use-cases, and stakeholders a continuous traceability between requirements is established.

The concrete method *process modeling* describes a formal approach to the scenario-based design of reactive systems. A set of informal scenarios capturing the executions within the system and its environment is formalized via a structured, labeled transition system, which can be composed and analyzed in a modular fashion. Owing to its operational character, this specification can also be simulated and used to synthesize a first, logical system architecture implementing the specification. The translation of informal scenarios into a logical architecture is structured according to six iterative development steps, for which tool support is currently being implemented.

11.3 Summary

None of the methods presented in this section is able, on its own, to solve all the problems of Requirements Engineering. Nevertheless, each of the presented methods deals with some important Requirements Engineering aspect. If all the approaches already available in our research group are integrated, they would provide an almost complete coverage of Requirements Engineering, with only few areas missing, as shown in Figure 11.2. Thus, it is our primary goal, to integrate our independently developed approaches. The presented chapter shows how they fit together and what additional questions should be taken into account in order to completely cover the whole Requirements Engineering.

12 Resumee

12.1 Conclusion

The presented report showed the comprehensive RE framework at Software & Systems Engineering at Technische Universität München. The framework covers most topics that are important for industrial Requirements Engineering, which makes it a promising solution for practical applications. It addresses both methodological and technical issue, and thus caters both for methodologists' and analysts' needs. The framework is based on a meta model for artefacts that provides the basis for two domain-specific artefact models for business information systems (REMbIS, see Chap. 3) and embedded systems (REMsES, see Chap. 2), respectively.

There are a number of artefact-based and model-based concepts, methods and techniques that are integrated with and can be used on the foundation of the given artefact models. We cover textual requirements patterns in Chap. 4, computational linguistics in Chap. 5, formalization of functional requirements in Chap. 6, modeling of usage environments of interactive systems in Chap. 7, modeling of human-machine systems in Chap. 8, transition from system to subsystem in Chap. 9, and risk management in Chap. 10.

Our construction kit of compatible building blocks provides comprehensive guidance for many of the challenges in requirements engineering, for technical tasks as well as for methodical issues, which makes it a viable alternative to current industrial RE practices.

12.2 Outlook

The different approaches, methods and techniques that are so far considered in integrating requirements engineering activities at the Informatics chair for Software and Systems Engineering at the Technische Universität München, constitute only an initial selection. There is currently a couple of approaches under development that are important to an integrated requirements engineering approach. These approaches address topics such as process modeling and synthesis from processes to transition systems, modeling quality requirements, analysing problem domain to name just a few examples.

Bibliography

[Abadi and Lamport, 1994] Abadi, M. and Lamport, L. (1994). Decomposing specifications of concurrent systems. In *PROCOMET*, pages 327–340.

[Abrial et al., 1996] Abrial, J.-R., Börger, E., and Langmaack, H. (1996). The steam boiler case study: Competition of formal program specification and development methods. In Abrial, J.-R., Borger, E., and Langmaack, H., editors, *Formal Methods for Industrial Applications*, volume 1165 of *LNCS*. Springer.

[Alur and Grosu, 2004] Alur, R. and Grosu, R. (2004). Modular refinement of hierarchic reactive machines. *ACM Trans. Program. Lang. Syst.*, 26(2):339–369.

[Benz, 2007] Benz, S. (2007). Combining Test Case Generation for Component and Integration Testing. In *A-MOST*, pages 23–33.

[Boehm, 1981] Boehm, B. W. (1981). *Software Engineering Economics*. Prentice-Hall.

[Boehm, 1991] Boehm, B. W. (1991). Software risk management: Principles and practices. *IEEE Softw.*, 8(1):32–41.

[Booch et al., 1999] Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *Das UML-Benutzerhandbuch*. Addison-Wesley.

[Braun et al., 2005] Braun, C., Wortmann, F., Hafner, M., and Winter, R. (2005). Method construction - a core approach to organizational engineering. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1295–1299. ACM.

[Brinkkemper, 1996] Brinkkemper, A. (1996). Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology*, pages 275–280.

[Broy, 2003] Broy, M. (2003). Service-oriented systems engineering: Modeling services and layered architectures. In *FORTE*, pages 48–61.

[Broy, 2005] Broy, M. (2005). Service-oriented systems engineering: Specification and design of services and layered architectures. In *Engineering Theories of Software Intensive Systems*, pages 47–81. Springer-Verlag.

[Broy, 2006] Broy, M. (2006). Challenges in automotive software engineering. In Osterweil, L. J., Rombach, H. D., and Soffa, M. L., editors, *ICSE*, pages 33–42. ACM.

[Broy et al., 2007] Broy, M., Cengarle, M. V., and Rumpe, B. (2007). Semantics of UML, Towards a System Model for UML, Part 3: The State Machine Model. Technical Report TUM-I0711, Technische Universität München.

[Broy et al., 2009] Broy, M., Leuxner, C., Sitou, W., Spanfelner, B., and Winter, S. (2009). Formalizing the Notion of Adaptive System Behavior. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1029–1033. ACM.

[Broy and Rumpe, 2007] Broy, M. and Rumpe, B. (2007). Modulare hierarchische modellierung als grundlage der software- und systementwicklung. *Informatik Spektrum*, 30(1):3–8.

[Buhr et al., 2004] Buhr, K., Heumesser, N., Houdek, F., Omasreiter, H., Rothermehl, F., Tavakoli, R., and Zink, T. (2004). DaimlerChrysler demonstrator: System specification instrument cluster. `http://www.empress-itea.org/deliverables/D5.1_Appendix_B_v1.0_Public_Version.pdf`, accessed 11.01.2007.

[Cimatti et al., 2002] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark. Springer.

[Clarke and Emerson, 1982] Clarke, E. M. and Emerson, E. A. (1982). Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs*, volume 131 of *LNCS*. Springer.

[Conway, 1968] Conway, M. (1968). How do committees invent? *Datamation Journal*, pages 28–31.

[Curran et al., 2006] Curran, J. R., Clark, S., and Vadas, D. (2006). Multi-tagging for lexicalized-grammar parsing. In *21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Sydney, Australia, 17-21 July*.

[Dix, 1991] Dix, A. J. (1991). *Formal Methods for Interactive Systems*. Computer and People Series. Academic Press.

[Endres and Rombach, 2003] Endres, A. and Rombach, H. (2003). *A handbook of software and systems engineering: empirical observations, laws and theories*. Addison-Wesley.

[Falkenberg, 1976] Falkenberg, E. D. (1976). Concepts for Modelling Information. In *IFIP Working Conference on Modelling in Data Base Management Systems*.

[Feilkas et al., 2009a] Feilkas, M., Fleischmann, A., Hölzl, F., Pfaller, C., Scheidemann, K., Spichkova, M., and Trachtenherz, D. (2009a). A Top-Down Methodology for the Development of Automotive Software. Technical report, Technische Universität München, Garching, Germany.

[Feilkas et al., 2009b] Feilkas, M., Hölzl, F., Pfaller, C., Rittmann, S., Schätz, B., Schwitzer, W., Sitou, W., Spichkova, M., and Trachtenherz, D. (2009b). A Refined Top-Down Methodology for the Development of Automotive Software Systems – The SmartKeyEntry-System Case Study. Technical report, Technische Universität München.

[Fleischmann, 2008] Fleischmann, A. (2008). *Modellbasierte Formalisierung von Anforderungen für eingebettete Systeme im Automotive-Bereich*. Grin Academic.

[Friedrich et al., 2009] Friedrich, J., Hammerschall, U., Kuhrmann, M., and Sihling, M. (2009). *Das V-Modell XT - Für Projektleiter und QS-Verantwortliche kompakt und übersichtlich*. Number ISBN: 978-3-642-01487-1 in Informatik im Fokus. Springer, zweite edition. available at http://www.springer.com/computer/programming/book/978-3-642-01487-1.

[Geisberger et al., 2006] Geisberger, E., Broy, M., Berenbach, B., Kazmeier, J., Paulish, D., and Rudorfer, A. (2006). Requirements engineering reference model (rem). Technischer Bericht, Technische Universität Müunchen.

[Ghezzi et al., 1991] Ghezzi, C., Jazayeri, M., and Mandrioli, D. (1991). *Fundamentals of Software Engineering*. Prentice-Hall, Inc., Englewood Cliffs, NJ.

[Green, 1985] Green, M. (1985). Report on dialogue specification tools. In *Seeheim Workshop on User Interface Management Systems*, pages 9–20. Springer-Verlag.

[Halpin, 2001] Halpin, T. (2001). *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufmann.

[Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.

[Harrison and Timbleby, 1990] Harrison, M. and Timbleby, H. (1990). The role of formal methods in human-computer interaction. In *Formal methods in human-computer interaction*, pages 1–8. Cambridge University Press, New York, NY, USA.

[Henzinger, 2000] Henzinger, T. A. (2000). Masaccio: A formal model for embedded components. In *IFIP TCS*, pages 549–563.

[Herbsleb and Grinter, 1999] Herbsleb, J. D. and Grinter, R. E. (1999). Splitting the organization and integrating the code: Conway's law revisited. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 85–95, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Islam, 2009] Islam, S. (2009). Software development risk management model: a goal driven approach. In *ESEC/FSE Doctoral Symposium '09: Proceedings of the doctoral symposium for ESEC/FSE on Doctoral symposium*, pages 5–8, New York, NY, USA. ACM.

[Islam et al., 2009] Islam, S., Joarder, M. M. A., and Houmb, S. H. (2009). Goal and risk factors in offshore outsourced software development from vendor's viewpoint. In *ICGSE '09: Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering*, pages 347–352, Washington, DC, USA. IEEE Computer Society.

[Jeffries et al., 1991] Jeffries, R., Miller, J. R., Wharton, C., and Uyeda, K. (1991). User interface evaluation in the real world: a comparison of four techniques. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 119–124, New York, NY, USA. ACM.

[Jensen, 1996] Jensen, F. (1996). *An introduction of bayesian network*. UCL press, University college London.

[Kof, 2005] Kof, L. (2005). *Text Analysis for Requirements Engineering*. PhD thesis, Technische Universität München.

[Kof, 2007a] Kof, L. (2007a). Scenarios: Identifying missing objects and actions by means of computational linguistics. In *15th IEEE International Requirements Engineering Conference*, pages 121–130, New Delhi, India. IEEE Computer Society Conference Publishing Services.

[Kof, 2007b] Kof, L. (2007b). Treatment of Passive Voice and Conjunctions in Use Case Documents. In *Application of Natural Language to Information Systems*, volume 4592 of *LNCS*, pages 181–192, Paris, France. Springer.

[Kof, 2008] Kof, L. (2008). From Textual Scenarios to Message Sequence Charts: Inclusion of Condition Generation and Actor Extraction. In *16th IEEE International Requirements Engineering Conference*, pages 331–332, Barcelona, Spain. IEEE Computer Society Conference Publishing Services.

[Kof, 2009a] Kof, L. (2009a). Requirements Analysis: Concept Extraction and Translation of Textual Specifications to Executable Models. In *Application of Natural Language to Information Systems*, LNCS, Saarbrücken, Germany. Springer, to appear.

[Kof, 2009b] Kof, L. (2009b). Translation of Textual Specifications to Automata by Means of Discourse Context Modeling. In Glinz, M. and Heymans, P., editors, *Requirements Engineering: Foundation for Software Quality, 15th International Working Conference*, volume 5512 of *LNCS*, pages 197–211. Springer.

[Kof and Schätz, 2003] Kof, L. and Schätz, B. (2003). Combining aspects of reactive systems. In *Ershov Memorial Conference*, volume 2890 of *LNCS*, pages 344–349. Springer.

[Kontio, 2001] Kontio, J. (2001). *Software Engineering Risk Management: A Method, Improvement Framework, and Empirical Evaluation*. PhD thesis, Helsinki University of Technology.

[Mannion and Keepence, 1995] Mannion, M. and Keepence, B. (1995). Smart requirements. *SIGSOFT Softw. Eng. Notes*, 20(2):42–47.

[Mendez Fernandez and Kuhrmann, 2009] Mendez Fernandez, D. and Kuhrmann, M. (2009). Artefact-based requirements engineering and its integration into a process framework. Technischer Bericht, Technische Universität Müunchen.

[Mich et al., 2004] Mich, L., Franch, M., and Novi Inverardi, P. (2004). Market research on requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56.

[Nuseibeh, 2001] Nuseibeh, B. (2001). Weaving the software development process between requirements and architecture. In *STRAW*.

[Parnas, 1972] Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058.

[Paterno, 1999] Paterno, F. (1999). *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London, UK.

[Pearl, 1988] Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Rittmann, 2008] Rittmann, S. (2008). *A methodology for modeling usage behavior of multi-functional systems*. PhD thesis, Technische Universität München.

[Ropponen and Lyytinen, 2000] Ropponen, J. and Lyytinen, K. (2000). Components of software development risk: How to address them? a project manager survey. *IEEE Trans. Softw. Eng.*, 26(2):98–112.

[Schätz, 2006] Schätz, B. (2006). Building components from functions. *Electr. Notes Theor. Comput. Sci.*, 160:321–334.

[Schätz, 2008] Schätz, B. (2008). *Model-Based Development of Software Systems: From Models to Tools*. Habilitation, Technische Universität München.

[Schätz, 2007] Schätz, B. (September 19-21, 2007). Modular functional specification of reactive components. In *Proceedings of Formal Aspects of Component Systems FACS'07*.

[Schätz et al., 2002] Schätz, B., Pretschner, A., Huber, F., and Philipps, J. (2002). Model-Based Development of Embedded Systems. In *Advances in Object-Oriented Information Systems, Lecture Notes in Computer Science*, volume 2426, pages 298–311. Springer-Verlag.

[Sitou, 2009] Sitou, W. (2009). *Requirements Engineering kontextsensitiver Anwendungen*. PhD thesis, Technische Universität München.

[Sitou et al., 2009] Sitou, W., Leuxner, C., and Spanfelner, B. (2009). Disciplined Modeling of Usage Context. In *12th International Conference on Medical Image Computing and Computer Assisted Intervention - Workshop on Modeling and Monitoring of Computer Assisted Interventions - (MICCAI - M2CAI 2009)*.

[Sitou and Spanfelner, 2007] Sitou, W. and Spanfelner, B. (2007). Towards Requirements Engineering for Context Adaptive Systems. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, volume 2, pages 593–600.

[Thurner, 2004] Thurner, V. (2004). *Formal fundierte Modellierung von Geschäftsprozessen*. PhD thesis, Technische Universität München.

[van Lamsweerde, 2009] van Lamsweerde, A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley.

[Wagner, 2006] Wagner, S. (2006). A Literature Survey of the Quality Economics of Defect-Detection Techniques. In *Proc. 5th ACM-IEEE International Symposium on Empirical Software Engineering (ISESE'06)*. ACM Press.

[Winter, 20XX] Winter, S. (20XX). *Modellbasierte Analyse von Nutzerschnittstellen*. PhD thesis, Technische Universität München. To appear.

[Wojcik et al., 2006] Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R., and Wood, B. (2006). Attribute-driven design (ADD). Technical Report CMU/SEI-2006-TR-023, CMU SEI Pittsburgh.