Gesellschaft für Informatik e.V. (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into
• seminars
• proceedings
• dissertations
• thematics
current topics are dealt with from the vantage point of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure high quality contributions.

The volumes are published in German or English.

Information: http://www.gi-ev.de/service/publikationen/lni/

ISSN 1617-5468
ISBN 978-3-88579-278-9

This volume contains papers from the workshops of the Software Engineering 2011 conference held in Karlsruhe from February 21st to 25th 2011. The topics covered in the papers range from enterprise engineering, software architecture, software-intensive embedded systems, software quality management, product line engineering to software evolution and software systems for emergency management.

Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.):
Software Engineering 2011 – Workshopband

184

GI-Edition

Lecture Notes in Informatics

Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.)

Software Engineering 2011 Workshopband

(inkl. Doktorandensymposium)

21.–25. Februar 2011 Karlsruhe

Proceedings



Ralf Reussner, Alexander Pretschner, Stefan Jähnichen
(Hrsg.)

Software Engineering 2011 – Workshopband

(inkl. Doktorandensymposium)

Fachtagung des GI-Fachbereichs Softwaretechnik

21. – 25. Februar 2011
in Karlsruhe

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-184

ISBN 978-3-88579-278-9
ISSN 1617-5468

Volume Editors
Ralf Reussner
Alexander Pretschner
Karlsruher Institut für Technologie (KIT)
Fakultät für Informatik
76128 Karlsruhe
Email: {reussner, alexander.pretschner}@kit.edu
Stefan Jähnichen
Technische Universität Berlin
Fakultät IV: Softwaretechnik
Ernst-Reuter-Platz 7
10587 Berlin
Email: jaehn@cs.tu-berlin.de

Series Editorial Board
Heinrich C. Mayr, Universität Klagenfurt, Austria (Chairman, mayr@ifit.uni-klu.ac.at)
Hinrich Bonin, Leuphana-Universität Lüneburg, Germany
Dieter Fellner, Technische Universität Darmstadt, Germany
Ulrich Flegel, SAP Research, Germany
Ulrich Frank, Universität Duisburg-Essen, Germany
Johann-Christoph Freytag, Humboldt-Universität Berlin, Germany
Thomas Roth-Berghofer, DFKI, Germany
Michael Goedicke, Universität Duisburg-Essen, Germany
Ralf Hofestädt, Universität Bielefeld, Germany
Michael Koch, Universität der Bundeswehr, München, Germany
Axel Lehmann, Universität der Bundeswehr München, Germany
Ernst W. Mayr, Technische Universität München, Germany
Sigrid Schubert, Universität Siegen, Germany
Martin Warnke, Leuphana-Universität Lüneburg, Germany

Dissertations
Dorothea Wagner, Karlsruher Institut für Technologie (KIT), Germany
Seminars
Reinhard Wilhelm, Universität des Saarlandes, Germany
Thematics
Andreas Oberweis, Karlsruher Institut für Technologie (KIT), Germany

© Gesellschaft für Informatik, Bonn 2011
printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Die Tagung *Software Engineering 2011* (SE 2011) wurde vom 21. bis 25. Februar 2011 als nunmehr siebte Veranstaltung einer inzwischen etablierten Reihe von Fachtagungen durchgeführt, deren Ziel die Zusammenführung und Stärkung der deutschsprachigen Software-Technik ist. Die SE 2011 bot ein Forum zum intensiven Austausch über praktische Erfahrungen, wissenschaftliche Erkenntnisse sowie zukünftige Herausforderungen bei der Entwicklung von Softwareprodukten bzw. Software-intensiven Systemen. Sie richtete sich gleichermaßen an Teilnehmer aus Industrie und Wissenschaft.

Die Software-Engineering-Tagungsreihe wird vom Fachbereich Software-Technik der Gesellschaft für Informatik e.V. getragen. Die Software Engineering 2011 wurde von den Software-Technik-Professoren des Karlsruher Instituts für Technologie (KIT) und des Forschungszentrum Informatik (FZI) veranstaltet.

Vor dem Hauptprogramm der Konferenz, das aus begutachteten Forschungsarbeiten und eingeladenen wissenschaftliche Vorträgen bestand, fanden sechs Workshops sowie vier Tutorials zu aktuellen, innovativen und praxisrelevanten Themen im Software Engineering statt. Weiterhin wurde wie im letzten Jahr ein Doktorandensymposium abgehalten. Die vielversprechenden und interessanten Themenvorschläge der eingeladenen jungen Forscher sind ebenfalls in diesem Band abgedruckt. Der vorliegende Workshopband wurde im Anschluss an die Tagung erstellt, sodass die Organisatoren die Möglichkeit hatten, Anregungen aus Diskussionen aus den Workshops aufzunehmen. Mit Ausnahme des Workshops *Software-Qualitätsmodellierung und -bewertung (SQMB)*, der einen eigenen Proceedingsband veröffentlicht, umfasst dieser Band sämtliche Workshops.

Die Durchführung der Tagung *Software Engineering 2011* wäre ohne die Mitwirkung der Sponsoren und vieler engagierter Personen nicht durchführbar gewesen. Ich bedanke mich bei allen Sponsoren der SE 2011. Darüberhinaus gebührt besonderer Dank Matthias Grund (andrena objects AG) und meinen Kollegen Andreas Oberweis für die Planung des Industrietags, Walter Tichy für die Organisation des Nachwuchswissenschaftlerprogrammes, Stefan Jähnichen (TU Berlin) für die Koordination des Workshop- und Tutorialprogramms, Alexander Pretschner für die Organisation des Doktorandensymposiums, Barbara Paech (Universität HD) und Theo Heußer (Bergstrassen-Gymnasium) für die Organisation des Informatiklehrer/-Innen-Tages und Dr. Mircea Trifu (FZI) für die Planung und Durchführung des SE-FIT. Ganz besonders bedanke ich mich bei meinem Mitarbeiter Jörg Henß für seinen unermüdlichen Einsatz rund um die Organisation der Tagung, sowie bei meinen Mitarbeitern Zoya Durdik, Erik Burger, Qais Noorshams, Andreas Rentschler und Benjamin Klatt, meiner Assistentin Tatiana Rhode und meiner Sekretärin Elena Kienhöfer für die große Hilfe bei der Vorbereitung, sowie bei allen Mitarbeitern und Studierenden meiner Forschungsgruppe für die große Unterstützung während des Ablaufs der Tagung. Ohne diese Unterstützung wäre die SE 2011 nicht möglich gewesen.

Karlsruhe, im Februar 2011 — Ralf Reussner, Tagungsleiter

Tagungsleitung
Ralf Reussner, KIT und FZI Karlsruhe

Leitung Industrietag
Wilhelm Schäfer, Universität Paderborn

Leitung Workshops und Tutorials
Stefan Jähnichen, TU Berlin

Tagungsorganisation
Matthias Grund, andrena objects AG
Wilfried Juling, KIT Karlsruhe
Andreas Oberweis, KIT und FZI Karlsruhe
Alexander Pretschner, KIT Karlsruhe
Ralf Reussner, KIT und FZI Karlsruhe
Stefan Tai, KIT Karlsruhe
Walter F. Tichy, KIT Karlsruhe

Programmkomitee
Steffen Becker, Universität Paderborn
Klaus Beetz, Siemens AG
Manfred Broy, TU München
Bernd Brügge, TU München
Jürgen Ebert, Universität Koblenz-Landau
Gregor Engels, Universität Paderborn
Martin Glinz, Universität Zürich
Michael Goedicke, Universität Duisburg-Essen
Volker Gruhn, Universität Leipzig
Jens Happe, SAP Research
Wilhelm Hasselbring, Christian-Albrechts-Universität zu Kiel
Stefan Jähnichen, TU Berlin
Matthias Jarke, RWTH Aachen
Gerti Kappel, TU Wien
Udo Kelter, Universität Siegen
Jens Knoop, TU Wien
Heiko Koziolek, ABB
Claus Lewerentz, BTU Cottbus
Horst Lichter, RWTH Aachen
Peter Liggesmeyer,TU Kaiserslautern
Oliver Mäckel, Siemens AG
Florian Matthes, TU München
Oscar Nierstrasz, Universität Bern
Andreas Oberweis, KIT und FZI Karlsruhe
Barbara Paech, Universität Heidelberg
Klaus Pohl, Universität Duisburg-Essen

Alexander Pretschner, TU Kaiserslautern
Ralf Reussner, KIT und FZI Karlsruhe
Matthias Riebisch, TU Ilmenau
Andreas Roth, SAP AG
Bernhard Rumpe, RWTH Aachen
Wilhelm Schäfer, Universität Paderborn
Klaus Schmid, Universität Hildesheim
Kurt Schneider, Leibniz Universität Hannover
Andy Schürr, TU Darmstadt
Rainer Singvogel, msg systems ag, München
Stefan Tai, KIT und FZI Karlsruhe
Walter F. Tichy, KIT und FZI Karlsruhe
Markus Voß, Accso – Accelerated Solutions GmbH, Darmstadt
Andreas Winter, Universität Oldenburg
Mario Winter, Fachhochschule Köln
Uwe Zdun, Universität Wien
Andreas Zeller, Universität des Saarlandes
Heinz Züllighoven, Universität Hamburg
Albert Zündorf, Universität Kassel

Offizieller Veranstalter
Fachbereich Softwaretechnik der Gesellschaft für Informatik (GI)

Mitveranstalter
Karlsruher Institut für Technologie (KIT)
Forschungszentrum Informatik Karlsruhe (FZI)

Unterstützt wird die Tagung zudem von
Schweizer Informatik Gesellschaft (SI)
Österreichische Computer Gesellschaft (OCG)

Sponsoren

SE 2011 Platinsponsoren















SE 2011 Goldsponsoren





SE 2011 Silbersponsoren



SE 2011 Bronzesponsoren



Inhaltsverzeichnis

Dritter Workshop zu „Design For Future – Langlebige Softwaresysteme“ (DFF 2011)

Dritter Workshop zu „Design for Future (DFF 2011) – Workshop Report“
Christof Momm, Stefan Sauer, Mircea Trifu 3

Evolution Service-Orientierter Systeme
Harry M. Sneed 9

Towards the Combination of Clustering-based and Pattern-based Reverse Engineering Approaches
Oleg Travkin, Markus von Detten, Steffen Becker 23

Kompatibilitätsanalyse bei Evolution framework-basierter Anwendungen
Fabian Christ, Jan-Christopher Bals 29

Enriching OSGi Service Interfaces with Formal Sequential Contracts
Marco Müller, Moritz Balz, Michael Goedicke 41

Towards Software Sustainability Guidelines for Long-living Industrial Systems
Heiko Koziolek, Roland Weiss, Zoya Durdik, Johannes Stammel, Klaus Krogmann ... 47

Introducing Adaptivity to Achieve Longevity for Software
Mahdi Derakhshanmanesh, Jürgen Ebert, Mehdi Amoui, Ladan Tahvildari 59

Zweiter Workshop zur Zukunft der Entwicklung softwareintensiver, eingebetteter Systeme (ENVISION2020)

ENVISION 2020: Zweiter Workshop zur Zukunft der Entwicklung softwareintensiver, eingebetteter Systeme
Kim Lauenroth, Mark Rzepka 73

A Seamless Model-Based Development Process for Automotive Systems
Jörg Holtmann, Jan Meyer, Matthias Meyer 79

Timing Overhead Analysis for Fault Tolerance Mechanisms
Kai Höfig 89

A pattern-based requirement specification language: Mapping automotive specific timing requirements
Philipp Reinkemeier, Ingo Stierand, Philip Rehkop, Stefan Henkler 99

Herausforderungen für das Requirements Engineering in der Domäne Energie
Friedrich-W. Fasse, Kim Lauenroth 109

Modellgetriebene Validierung von System-Architekturen
André Pflüger, Wolfgang Golubski, Stefan Queins 119

Evolutionäre Software- und Systementwicklung – Methoden und Erfahrungen (ESoSyM-2011)

Evolutionäre Software- und Systementwicklung - Methoden und Erfahrungen (ESoSyM-2011)
Matthias Riebisch, Gregor Engels, Bernhard Schätz, Christian Zeidler 131

On the Impact of Goals on Long-Living Systems
Arfan Mansoor, Detlef Streitferdt 133

A Qualitative Discussion of Different Approaches for Implementing Multi-Tenant SaaS Offerings1
Christof Momm, Rouven Krebs 139

Traceability Link Evolution with Version Control
Patrick Mukherjee, Karsten Saller, Aleksandra Kovacevic, Kalman Graffi, Andy Schürr, Ralf Steinmetz 151

Komponentenbasierte Entwicklung von Firmware für Sensorsysteme in der industriellen Praxis
Philipp Nenninger, Tilo Merlin, Dirk Joh, Florian Kantz* 163

IT-Unterstützung von Einsatz- und Rettungskräften: Interdisziplinäre Anforderungsanalyse, Architekturen und Gestaltungskonzepte

Workshop „IT-Unterstützung von Einsatz- und Rettungskräften: Interdisziplinäre Anforderungsanalyse, Architekturen und Gestaltungskonzepte“
Frank Fiedrich, Birgitta König-Ries, Rainer Koch, Volkmar Pipek, Stefan Strohschneider 177

FRIENDS – eine Kommunikationsbrücke zwischen informatikfremden Fachexperten und Softwareentwicklern in der Anforderungsermittlung
Ivonne Erfurth und Kathrin Kirchner 181

Interdisziplinäre Anforderungsanalyse bei der Gestaltung von Systemen zur Erstellung eines gemeinsamen Lagebilds in großen Schadenslagen
Christian Lindemann, Jens Pottebaum, Gesine Hofinger, Laura Künzer, Rainer Koch . 193

Requirements Engineering: Ein Ansatz auch für die klassische Produktentwicklung?
Christian Lindemann, Kurt-Robert Hippler, Rainer Koch 205

Effiziente Entscheidungsunterstützung im Krisenfall durch interaktive Standard Operating Procedures
T. Becker, B.-S. Lee, R. Koch 217

A Service-oriented Architecture for Emergency Management Systems
Stephan Braune, Achim D. Brucker, Gerald Kleser, Keqin Li, Theo Dirk Meijler, Heiko Paulheim, and Florian Probst 225

Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation (PIK2011)

Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation – Workshop-Einleitung und Überblick
Andreas Birk, Klaus Schmid, Markus Völter 235

MOD2-SCM: Eine modellgetriebene Produktlinie für Softwarekonfigurationsverwaltungssyteme
Thomas Buchmann, Alexander Dotor 241

Standardschnittstellen als nichtfunktionale Variationspunkte: Erfahrungen aus der EPM-Produktlinie
Martin Frenzel, Jörg Friebe, Simon Giesecke, Till Luhmann 253

A Formal Method to Identify Deficiencies of Functional Requirements for Product Lines of Embedded Systems
Florian Markert, Sebastian Oster 265

Doktorandensymposium

Online Performance Prediction with Architecture-Level Performance Models
Fabian Brosig 279

A Proposal on Validation of an Agile Architecture-Modelling Process
Zoya Durdik 285

Identifikation der Potenziale dreidimensionaler Softwarevisualisierung – Zielstellung und Vorgehen
Richard Müller 291

Wertorientierte Portfolio-Optimierung bei Software-Produktlinien
Johannes Müller 297

Semi-automatische Ableitung externer Anpassungsmechanismen für selbst-adaptive Systeme
Benjamin Nagel 303

Model Synchronization for Mechatronic Systems
Jan Rieke 309



Dritter Workshop zu „Design For Future – Langlebige Softwaresysteme“ (DFF)

Design for Future (DFF 2011) – Workshop Report

Christof Momm[1], Stefan Sauer[2], Mircea Trifu[3]

[1] SAP Research, Karlsruhe, Germany
christof.momm@sap.com
[2] Universität Paderborn, s-lab – Software Quality Lab, Paderborn, Germany
sauer@s-lab.upb.de
[3] Forschungszentrum Informatik (FZI), Karlsruhe, Germany
GI-Arbeitskreis „Langlebige Softwaresysteme“ (AK L2S2)
der Fachgruppen „Software-Architektur“ und „Software-Reengineering“
(http://akl2s2.ipd.kit.edu/)
mtrifu@fzi.de

Abstract: Software altert und erodiert, obwohl sie ein immaterielles Gut ist. Denn sowohl die Anforderungen an die Software als auch ihre Umgebung ändern sich mit der Zeit. Bei Informationssystemen ist dieses Phänomen als „Legacy“ wohlbekannt. Das Problem existiert aber auch bei eingebetteten Systemen, wo komplexe Software in langlebigen technischen Geräten eingesetzt wird. Die ökonomischen Auswirkungen der Software-Alterung sind beträchtlich. Wissenschaft und Industrie sind deshalb aufgefordert, neue Methoden der Softwaretechnik für zukunftsfähige Software zu entwickeln und die erheblichen Investitionen in große Softwaresysteme zu schützen. Beim 3. Workshop „Design for Future“ des Arbeitskreises „Langlebige Softwaresysteme“ (L2S2) diskutierten Wissenschaftler und Praktiker Herausforderungen, Erfahrungen und Lösungen aus praktischer wie aus wissenschaftlicher Sicht.

1 Einleitung

Aktuelle Ansätze der Softwaretechnik wie modellbasierte Entwicklungsmethoden, Softwarearchitektur, Evolution und Lifecycle-Management, Qualitätsmanagement und Software-Reengineering können dazu beitragen, die Herauforderungen der Software- Alterung anzugehen und die Situation zu verbessern, wenn sie geeignet weiterentwickelt und angewandt werden. Für den 3. Workshop „Design for Future“, der im Rahmen der Konferenz „Software Engineering 2011“ stattfand, wurden vier Themenblöcke als inhaltliche Schwerpunkte im *Call for Papers* vorgegeben:

- Integrierte und modellgetriebene Entwicklungsmethoden für langlebige Software
- übergreifendes Application-Lifecycle-Management
- Anpassungsfähige und zukunftssichere Software-Architekturen
- Qualitätsmanagement

2 Ablauf des Workshops

Der Workshop „Design for Future 2011“ gliederte sich in zwei wesentliche Teile. Im ersten Teil wurden aktuelle Fragestellungen und Lösungen aus Wissenschaft und Praxis in Vorträgen präsentiert und ausgiebig diskutiert. Eröffnet wurde der Workshop mit einem eingeladenen Vortrag. Im Anschluss präsentierten die Autoren der akzeptierten Workshopbeiträge ihre Arbeiten. Im zweiten Teil wurden Themen, Ziele und Fragestellungen einer Forschungsagenda für langlebige Softwaresysteme aus drei thematischen Perspektiven – modellgetriebene Entwicklungsmethoden, Application- Lifecycle-Management, Software-Architektur- und Qualitätsmanagement – in parallelen Kleingruppen diskutiert, die abschließend ihre Ergebnisse im Plenum vorstellten.

3 Vorträge und Diskussionen

In seinem eingeladenen Vortrag bediente sich Dr. Axel Uhl, *Chief Development Architect* der SAP AG, der Analogie des alternden Kartoffelsalates um in sehr anschaulicher und unterhaltsamer Weise die Probleme der Software-Alterung und Mittel dagegen herauszustellen. Basierend auf seinen Erfahrungen mit großen betrieblichen Informationssystemen identifizierte er drei wesentliche Gründe für die Alterung von Software: (1) sich ändernde Anforderungen an die Software seitens der Kunden bzw. des Marktes, (2) die Weiterentwicklung/Ablösung der verwendeten Plattformen sowie (3) die unbedachte Wiederverwendung bestehender Komponenten bei der Weiterentwicklung des Systems.

Als wesentliches Mittel gegen solche schädlichen Einflüsse sieht er die Schaffung zeitloser Designs auf Grundlage guter Abstraktionen für die entsprechende Domäne. Geeignete Modelle haben seiner Ansicht nach wesentlich dazu beitragen, solche Designs über Generationen sich ändernder Plattformen und Anforderungen hinweg zu transportieren. Darüber hinaus bedarf es natürlich einer kontinuierlichen Pflege des Systems, welche durch wohldefinierte *Governance*-Prozesse gesteuert wird. Dies hängt jedoch stark von der Wertschätzung des durch die Software gelieferten „Inhalts“ zusammen. Bricht die *Community* weg, ist die Alterung und letztendlich Ausmusterung der Software vorprogrammiert.

Im Anschluss an den eingeladenen Vortrag ging Harry Sneed in seinem Vortrag der Frage nach, welche Eigenschaften von Software-Services den Evolutionsprozess beeinflussen oder beeinträchtigen. Als wesentliche Hindernisse identifizierte er (1) Mehrsprachigkeit der Software, (2) Schnittstellenkomplexität und (3) Lieferantenabhängigkeit – und stellte ihnen geeignete Maßnahmen gegenüber: mehrsprachige Werkzeuge und Mitarbeiter, strenge Konventionen, saubere Architekturen, harte, aber faire Service Level Agreements sowie redundante Services.

In der zweiten Vortragsrunde stellte Heiko Koziolek ein Regelwerk für die nachhaltige Entwicklung langlebiger softwareintensiver Systeme in der industriellen Automatisierung auf Basis wiederkehrender Evolutionsszenarien vor. Das Regelwerk ist das Ergebnis einer umfassenden Literaturrecherche und soll nun im praktischen Einsatz evaluiert werden. Ziel des von Fabian Christ vorgestellten Ansatzes zur Kompatibilitätsanalyse frame-

workbasierter Anwendungen ist es, die Kompatibilität einer Anwendung zu einer neuen Frameworkversion ohne eine „Migration auf Probe“ zu bewerten. Stattdessen wird ein automatisierter, dreistufiger Prozess auf Grundlage eines Metamodells für Framework-Beschreibungssprachen vorgeschlagen. Der Prozess aus Benutzungsanalyse, Differenzanalyse und Kompatibilitätsanalyse liefert Hinweise auf Kompatibilitätsverletzungen. Die Kompatibilität von OSGi Service-Schnittstellen ist Gegenstand des Ansatzes, den Marco Müller vorstellte. Mit Hilfe sequentieller Kontrakte, die als Zustrandsautomaten formalisiert sind, soll die korrekte Interaktion von Komponenten beschrieben, analysiert und durchgesetzt werden.

Die Nutzung eines graphbasierten Adaptivitäts-Frameworks im Kontext der Software-Evolution stellte Mahdi Derakhshanmanesh in der dritten Vortragsrunde vor. Neben der selbst-adaptiven Anpassung an kleinere vorhergesehene änderungen (Micro Adaptation) wird so auch die Anpassung der Software an größere, unvorhergesehene änderungen im Rahmen von Wartungsaktivitäten (Macro Adaptation) unterstützt. Das Framework nutzt ein Laufzeitmodell der anpassungsfähigen Software und eine Anpassungs-Middleware, um die Software zur Laufzeit mittels Zustandsvariablen und Anpassungsregeln zu überwachen und anzupassen. Macro Adaptation besteht dann in der Veränderung der Regelmenge. Die Kombination von Clustering- und Pattern-basierten Reverse-Engineering-Ansätzen empfahl Markus von Detten in seinem Vortrag. Mit Hilfe des Clustering soll die Systemarchitektur zurück gewonnen werden, sogenannte „Bad Smells“ können aber die Clustering-Ergebnisse verfälschen. Deshalb sollen „Bad Smells“ zuvor mittels Mustererkennung entdeckt und entfernt werden, um so die Qualität der Architektur zu verbessern.

Im Anschluss an die Präsentation der akzeptierten Workshop-Papiere stellte Prof. Dr. Ursula Goltz das beantragte Schwerpunktprogramms „Design for Future – Managed Software Evolution“ vor.

4 Ergebnisse der Arbeitsgruppen

Ziel der abschließenden Diskussionsgruppen war es, wesentliche Forschungsfragen im Kontext der Entwicklung, Evolution und des Betriebs langlebiger Softwaresysteme zu identifizieren und herauszustellen. Damit sollte der Grundstein für eine Forschungsagenda gelegt werden, die im Rahmen weiterer Arbeitstreffen sukzessive weiter ausgearbeitet werden soll.

Die Einteilung in Diskussionsgruppen erfolgte auf Basis der inhaltlichen Schwerpunktbildung, die sich an den im *Call for Papers* aufgeführten Themenblöcken orientierte: modellgetriebene Entwicklungsmethoden (MDD), Application-Lifecycle-Management sowie Software-Architektur- und Qualitätsmanagement.

Um die Zusammenführung der jeweils erarbeiteten Einzelergebnisse zu erleichtern, wurde das folgende Schema – angelehnt an den „Goal-Question-Metric“-Ansatz – für die Erarbeitung und Erfassung der Forschungsfragestellungen vorgeschlagen.

1. **Ziele** aufstellen: Herausstellen der wesentlichen Ziele für den jeweiligen inhaltli-

chen Schwerpunkt.

2. **Fragestellungen** ableiten: Für jedes aufgeführte Ziel werden wesentliche Forschungsfragen identifiziert, die zur Erreichung des Ziels beantwortet werden müssen.
3. **Bestehende Technologien/Forschungsansätze** erfassen: Für jede Fragestellung werden (soweit bekannt) bereits bestehende Technologien oder Forschungsansätze aufgeführt, die potenziell zur Beantwortung der jeweiligen Forschungsfragestellung beitragen.

Die folgenden Abschnitte fassen die Ergebnisse der drei Diskussionsgruppen zusammen. Aus Platzgründen beschränken wir uns in diesem Bericht auf eine Auswahl der wichtigsten Ziele und Fragestellungen.

4.1 Modellgetriebene Entwicklungsmethoden (MDD)

Die Diskussionsgruppe zum Thema „Modellgetriebene Entwicklungsmethoden“ vereinte praktische und wissenschaftliche Perspektiven. Das erste zentrale Ziel war die Identifikation der richtigen Modelle für langlebige Softwaresysteme. Als typische Fragestellungen in diesem Zusammenhang identifizierte die Gruppe:

- Auf welcher Granularität modellieren wir?
- Für welchen Zweck werden die Modelle erstellt?
- Welche Aspekte werden modelliert?
- Wie kann die Angemessenheit der Modellierung evaluiert werden?
- Was ist die geeignete Repräsentation eines Modells?

Lösungsansätze können Domänenspezifische Sprachen (DSL) und die systematische Entwicklung von Modellierungssprachen (Language Engineering) liefern. Ein weiteres Ziel, das betrachtet wurde, ist der übergreifende Einsatz modellbasierter Methoden für Entwicklung und Betrieb. Hierbei sind unter anderem die Fragen zu beantworten:

- Welche Modelle braucht man für den Betrieb?
- Welche Entwicklungsmodelle können im Betrieb genutzt werden?

Techniken, die für eine Annäherung an die Lösung genutzt werden können, sind beispielsweise Design-by-Contract-Modelle, Performance-Modelle, Adaptionsmodelle. Die Evolutionsfähigkeit von Metamodellen und Modeltransformationen stellt ein weiteres wichtiges Ziel für die Langlebigkeit modellgetriebener Entwicklungsmethoden dar. Typische Fragestellungen sind:

- Wie kann die Gültigkeit von Modellen nach der Evolution geprüft werden?

- Wie erhält man die Traceability von Transformationen zu Metamodell-Versionen?
- Wie langlebig sind Modellierungssprachen?
- Wie kann ich gewährleisten, dass der Technologie-Stack noch funktioniert?

Benötigt werden hierzu geeignete Richtlinien im Umgang mit MDD-Techniken. Weitere Ziele, die diskutiert und mit konkreten Fragestellungen unterlegt wurden, waren Variabilitätsmanagement, insbesondere auf Architekturebene, Roundtrip-Engineering und die Co-Evolution von Modellen (Anforderungen, Architektur) und Implementierung sowie das Management, die Evolution und die Versionierung von Modellen. Auch zu diesen Fragen konnte die Gruppe erste potenzielle Lösungsansätze identifizieren.

4.2 Application-Lifecycle-Management

Die Diskussionsgruppe zum Thema „Application-Lifecycle-Management“ (ALM) hatte eine starke Industriebeteiligung aus unterschiedlichen Domänen: Prozessleitsysteme, Automotive (Embedded) Systeme sowie betriebliche Informationssysteme. Der erste Schritt bestand daher in der Klärung des Aufgabenbereichs und der Herausstellung der wesentlichen Herausforderungen. Unabhängig von der Domäne wurden Installation, Upgrade, Update und ein übergreifendes Versionsmanagement als Kernaufgaben des ALM identifiziert. Bei weiterführenden ALM-Aufgaben, wie z.B. der überwachung und Analyse von Laufzeitinformationen oder der Steuerung bzw. den Selbstheilungsfähigkeiten des Systems, wichen dagegen die Anforderungen stark voneinander ab. Daher wurde in Hinblick auf die zu erarbeiteten Ziele und Fragstellungen der Fokus auf die Kernaktivitäten gelegt.

Das wichtigste Ziel im Rahmen der Kernaktivitäten ist nach Einschätzung aller Beteiligten dabei die Entkopplung de Lebenszyklen verschiedener, in unterschiedlichen Geschwindigkeiten co-evolvierender Systemteile, Schichten oder Bauteile. Daneben werden die weitergehende Automatisierung, Vorhersagbarkeit der Aufgaben bzgl. Dauer/Kosten, die Unterbrechungsfreiheit des Betriebs, die änderungstoleranz zur Laufzeit sowie die kontinuierliche Verbesserung als wichtige Ziele gesehen. Für jedes Ziel konnte bereits eine Reihe von Fragestellungen herausgearbeitet werden. So sind für eine Entkopplung der Lebenszyklen unter anderem die folgenden Forschungsfragen zu beantworten:

- Wie kann man den Impact von änderungen automatisiert ermitteln?
- Wie sollte ein Erweiterungskonzept für ALM-Funktionen aussehen?

Helfen können bei der ersten Fragestellung z.B. bestehende Techniken der Code- Analyse aus dem Bereich des Reverse-Engineering, aber auch Forschungsansätze zur Nachverfolgbarkeit in Modellen (Model Traceability).

4.3 Software-Architektur- und Qualitätsmanagement

Die Diskussionsgruppe „Software-Architektur- und Qualitätsmanagement“ hatte eine maßgebliche Beteiligung von Akademikern. Die Beteiligten setzten die Evolution von Software-Architekturen als zentrales Ziel in den Mittelpunkt. Dabei wurden zwei Unteraspekte besonders herausgestellt: das Management und die Qualitätssicherung bei der Evolution von Software-Architekturen.

Die aus diesen Zielen herausgearbeiteten wichtigsten Forschungsfragen können in drei Gruppen gebündelt werden:

- Inwieweit ist Langlebigkeit planbar/vorhersehbar und welche typischen Evolutionsszenerien gibt es?
- Wie erreicht man Langlebigkeit bei alten Systemen?
- Wie misst man Langlebigkeit? Was sind typischen Qualitätskriterien/ Qualitätsmetriken? Wie sieht der Trade-off zwischen Langlebigkeit und anderen Software-Eigenschaften (Kosten, Performance) aus?

Helfen können bei der ersten Fragengruppe bestehende Analyseansätze wie Change-Impact- und Traceability-Analysen, aber auch eine stärkere Wiederverwendung des Erfahrungswissens in Form von Produktlinien, Referenzarchitekturen und Architekturmustern. Bei der zweiten Gruppe wurden Reengineering- und Migrationsansätze im Vordergrund gesehen, und bei der letzten Gruppe ist eine Mischung von manuellen Architektur-Reviews und automatisierten Mess- und Verifikationsverfahren geeignet, die verschiedene Qualitätsstandards, Randbedingungen und Szenarien mit Hilfe von Metriken und Heuristiken überprüfen sollen.

5 Zusammenfassung und Ausblick

Der diesjährige Workshop DFF 2011 zählte mit qualitativ hochwertigen Beiträgen, gehaltvollen Diskussionen, durchgängig über 30 Teilnehmer/innen und der allgemein positiven Resonanz zu den erfolgreichsten Workshops der Konferenz „Software Engineering (SE) 2011“. Wir bedanken uns daher nochmals bei allen Autoren/innen der Workshop-Beiträge, den Teilnehmer/innen sowie den Vortragenden sehr herzlich für die spannenden Beiträge und Diskussionen und bei den Mitgliedern des Programmkomitees für die Begutachtung der eingereichten Beiträge, ohne die ein solcher Erfolg nicht möglich gewesen wäre. Besonderer Dank gilt aber auch dem Team der SE 2011 für die hervorragende Organisation der Konferenz sowie den studentischen Hilfskräften, die uns bei der Durchführung des Workshops tatkräftig zur Seite standen.

Die begonnene Arbeit an der Forschungsagenda soll nun bei folgenden Arbeitskreistreffen zu ausgewählten Themen fortgesetzt und weiter detailliert werden. Darüber hinaus ist geplant, die Workshop-Reihe „Design for Future“ im kommenden Jahr fortzusetzen, dann wieder gemeinsam mit dem „Workshop Software- Reengineering“ der gleichnamigen GI-Fachgruppe.

Evolution Service-Orientierter Systeme

Harry M. Sneed
ANECON GmbH
Alserstrasse 4, A-1090 Wien
Harry.Sneed@T-Online.de

Abstract: In diesem Beitrag geht es um die Evolution, bzw. Wartung und Weiterentwicklung, Service-orientierter Systeme (SoS). Zunächst werden die besonderen Eigenschaften solcher Systeme vor dem Hintergrund einer Service-orientierter Architektur(SoA) erläutert. Danach werden drei Probleme vorgestellt, die im Zusammenhang mit der Evolution Service-orientierter Systeme stehen: die Mehrsprachigkeit, die gestiegene Komplexität und die vermehrte Abhängigkeit von externen Lieferanten. Der Beitrag geht darauf ein, welchen Einfluss diese Probleme auf die Evolution der Software haben werden. Anschließend werden alternative Möglichkeiten aufgezeichnet mit diesen Problemen fertig zu werden.

Schlüsselbegriffe: Software-Evolution, Web-Services, service-orientierte Architekturen, Wartung, Weiterentwicklung, Reverse-Engineering, Regressionstest.

1. Software-Evolution

Der Begriff Software-Evolution wurde bereits in den 70er Jahren von Les Belady und Manny Lehman geprägt. Im Jahre 1976 erschien ihre erste Studie über die Evolution des OS-Betriebssystems bei der IBM [BL75]. Schon damals wurde festgestellt, dass Software sich wandeln muss wenn sie überleben will. Anderseits, je mehr sie sich wandelt, desto brüchiger wird sie. Die Komplexität steigt und die Qualität sinkt. Irgendwann lässt sich die Software nicht mehr verändern und muss ausgemustert werden [LB85]. Allerdings haben Belady und Lehman dieses Erosionsprozesses grob unterschätzt. Sie konnten damals nicht ahnen, dass er Jahrzehnte dauern könne. Sie haben auch die Möglichkeit zur Sanierung der Software unterschätzt. Durch regelmäßige Überholungen lässt sich die Lebensdauer eines Softwaresystems verlängern. Als sie ihre Gesetze der Software-Evolution formulierten, steckte die Reengineering-Technologie erst in den Kinderschuhen. Heute haben wir Reengineering-Techniken mit denen wir sowohl die Komplexität einer Software abbauen als auch die Qualität steigern können [Sn02].

Trotzdem behalten die Gesetze der Software-Technologie noch ihre Gültigkeit. Nichts auf dieser Welt ist unsterblich, auch Software nicht. Sie wird immer älter weil die Technologie in der sie implementiert ist, immer weiter hinter den neuen Technologien zurückfällt und keine Reengineering-Techniken der Welt kann sie erneuern [Sn04]. Ein in Java migriertes COBOL System bleibt in seinem Wesen ein COBOL System obwohl es in Java kodiert ist. Der Geist der COBOL Urväter lässt sich immer erkennen, nicht nur in den Datenstrukturen und in den Algorithmen, sondern gerade in der Gesamtarchitektur des Systems. Wir können also den Tod eines Softwaresystems nicht verhindern, aber wir können ihn durch geeignete Maßnahmen noch lange vor uns herschieben, vielleicht bis die zuständigen Mitarbeiter pensioniert sind. Dieser Autor hat beobachtet, dass Systemablösungen in der Regel mit einem Generationswechsel in einer Firma verbunden sind. Wenn die Alten gehen folgt ihnen ihre alte Software nach. Dies könnte als Ergänzung zu den Gesetzen der Software-Evolution angesehen werden. Ein Softwaresystem lebt nur so lange wie seine Väter.

Jedenfalls ist Software-Evolution heute ein fester Begriff, inzwischen auch in dem deutschsprachigen Raum. Seine Bedeutung geht über das hinaus was allgemein unter Softwarewartung verstanden wird. Softwarewartung impliziert die Erhaltung der Software in seiner ursprünglichen Form. Sie wird zwar korrigiert, adaptiert und optimiert, aber sie bleibt in ihrer Funktionalität wesentlich gleich [vH05]. Unter Software-Evolution verstehen wir hingegen Wartung und Weiterentwicklung. Die Software wird nicht nur erhalten, sondern ständig erweitert und ergänzt um neue Funktionalitäten und neue technische Eigenschaften. Sie kann sogar in eine neue Sprache und/oder in eine neue Umgebung versetzt werden. Wichtig dabei ist, dass ihr betrieblicher Wert für ihre Benutzer im Sinne der Werttheorie zunimmt.

Nach Bennett und Rajlich ist auch die Zeit für die Software-Evolution begrenzt. Irgendwann lohnt es sich nicht mehr in ein System weiter zu investieren, nämlich dann wenn die Kosten der Erweiterung den Nutzen übersteigen. In dem Moment tritt das System in eine Erhaltungsphase in der es nur noch korrigiert und angepasst wird. Diese Phase der Wartung kann eine Weile dauern, dann wird das System durch die technologische Weiterentwicklung überholt und muss abgelöst werden. Es lohnt sich auch nicht mehr das System zu erhalten [BR00].

2. Service-orientierte Systeme

Heute stehen wir vor einem erneuten Aufbruch in der Software-Technologie – der Übergang zu service-orientierten Systemen. Viele reden davon, einige Wenige haben ihn schon vollzogen. Hier stellt sich nun die Frage, was ist eigentlich ein Service-orientiertes System und wie unterscheidet es sich von bisherigen Systemen.

Ein Service-orientiertes System könnte man als ein System mit einer service-orientierter Architektur bezeichnen, aber das wäre zu kurz gegriffen. Denn die Architektur bestimmt den Rahmen innerhalb dessen das jeweilige System operiert. Eine SOA ist wie ein Mega-Betriebssystem, manche betrachten sie sogar als Betriebssystem des 21 Jahrhunderts [KBS05]. Sie fungiert als Plattform für die einzelnen Benutzerapplikationen. Ein System ist wiederum aus der Sicht seiner Benutzer eine einzelne Applikation wie z.B. ein Buchungssystem, ein Kontoführungssystem oder ein Lagerhaltungssystem. Das Applikationssystem benutzt nur die Architektur. Demnach ist ein Service-orientiertes System ein Applikationssystem, das in einer Service-orientierter Architektur eingebettet ist und Services, sprich Web-Services, benutzt [SL07].

Ein Software-Service ist eine Software-Komponente die eine allgemeingültige Funktion erfüllt, die mehrere andere Software-Komponente benutzen können [NL05]. Früher hat man sie im Deutschen als Dienstprogramme und im englischen als Utilities bezeichnet. Ein typisches Dienstprogramm war ein Sort, das ein Applikationsprogramm aufrief, um eine Menge Objekte nach einem bestimmten Kriterium zu sortieren. Das Program übergab dem Dienstprogramm eine ungeordnete Menge mit Duplikaten und bekam eine geordnete Menge ohne Duplikate zurück. Derartige software-technische Dienste sind schon seit dem Anfang der elektronischen Datenverarbeitung ein fester Bestandteil fast aller Applikationssysteme. Es beschränkt sich auch nicht auf reine technische Dienste. Es wurden ebenfalls schon seit Beginn der Programmierung allgemeingültige fachliche Dienste benutzt. Ein hervorragendes Beispiel ist die FORTRAN mathematische Subroutinen-Bibliothek. Darin gab es für jede erdenkliche mathematische Funktion eine FORTRAN Subroutine, die über eine Call-Schnittstelle aufgerufen wurde. Das nutzende Programm gab die Argumente als Eingangsparameter vor und bekam das Ergebnis als Rückgabewert zurück. Allerdings musste man diese Bibliothek bestellen und in die eigene FORTRAN Umgebung einbinden.

Ähnliche Subroutinen-Bibliotheken gibt es auch für statistische und graphische Funktionen. In Objekt-orientierten Systemen erfüllen diese Rolle die Klassenbibliotheken [Lo93]. Es gibt Klassenbibliotheken für alles möglich – für statistische Funktionen, für mathematische Funktionen, für graphische Anzeigen und auch für fachliche Dienste wie Kalenderführung. Kein Programmierer würde auf die Idee kommen selber alles zu programmieren wenn sovieles vorprogrammiert vorliegt. Auch wenn er auf diese völlig irrationale Idee käme, würde er sich zunächst eine eigene Klassenbibliothek mit allgemeingültigen Klassen aufbauen um sich später die Arbeit zu erleichtern.

So gesehen ist das Service-Konzept absolut nichts Neues und verdient keine besondere Aufmerksamkeit. Was neu ist, ist wie die Dienste angeboten werden. Bisher waren die Dienste mit der jeweiligen Programmiersprache verbunden. Es gab eine FORTRAN Common Subroutine-Library oder die PL/I eingebaute Funktionen oder die COBOL Dienstprogramme. Später gab es C++ und Java Klassenbibliotheken. Es war nicht ohne weiteres möglich Dienste außerhalb des eigenen Sprachraums zu benutzen. Der erste Ansatz den Zugriff auf fremdsprachliche Dienstprogramme zu ermöglichen war die IBM Cross System Product Architektur im Rahmen der AD-Cycle Entwicklungsumgebung, aber dieser Ansatz war proprietär [Mo91]. Das IBM Systems Application Architecture (SAA) war ein Vorgriff auf SoA. Es folgte die Open Systems Foundation und die Common Object Request Broker Architektur – CORBA – die eine Verbindung über Sprach- und Betriebssystemgrenzen hinweg zugelassen hat [SC99]. Damit konnten C++ Programme auf einer UNIX Plattform auf COBOL Programme auf einem Mainframe-Rechner zugreifen. Dennoch hat aber auch dieser Ansatz noch nicht die Bezeichnung „Service-orientiert“ verdient.

Service-orientierte Systeme enthalten nicht nur allgemeingültige Dienstkomponente, sie sind auch noch verteilt. Die Services, bzw. die Server-Komponente, sind nicht auf dem gleichen Rechner wie die Client-Komponente. Sowohl die Server-Komponente als auch die Client-Komponente sind auf verschiedene Netzknoten verteilt. Verbunden werden sie miteinander über eine Middleware-Infrastruktur. In klassischen Client/Server Systemen war diese Middleware proprietär. Um seine verteilte Komponente miteinander zu verbinden, musste der Anwender ein Middleware-Produkt kaufen, ein sogenanntes Enterprise Application Integration Framework [Ke02]. Enterprise Application Integration – EAI – zielte darauf hin, eine Schicht von vorgefertigten Software Services, bzw. allgemeingültige Dienstkomponente, bereitzustellen [CHK06].

Inzwischen kann jeder über das Internet auf entfernte Web-Services zugreifen. Er muss nur eine Berechtigung haben. Man braucht keine aufwendige EAI-Infrastruktur, es sei denn für Sicherheitszwecke. Das World Wide Web ermöglichte den unbegrenzten Zugang zu entfernten Dienste am anderen Ende der Welt. Das Web wurde zum wichtigsten Instrument der Systemintegration und alles bewegte sich in diese Richtung [SnSn03]. Falls man aber einen entfernten Dienst in eine seiner Applikationen einbauen möchte, sollte man zuerst sichern, dass der entfernte Dienst immer verfügbar bleiben und immer die gleichen Ergebnisse liefern wird. Es ist prinzipiell möglich Web Services aus völlig unterschiedlichen Quellen in einer einzigen Service-orientierter Applikation einzubinden. Damit bekommen wir aber nicht nur ein Problem mit diversen Sprachen sondern noch ein Problem mit diversen Lieferanten.

Hinzu kommt ein drittes Problem auf uns zu und zwar die steigende Komplexität derartig hybriden Systeme. Komplexität wächst mit der Zahl der Beziehungen. Je mehr fremde Services in eine Applikation, bzw. in einen Geschäftsprozess, eingebaut werden, desto mehr Beziehungen wird es geben. Vor allem dann wenn viele Prozesse die gleichen Services verwenden, wird die Zahl der Netzbeziehungen relativ zur Zahl der Netzknoten ansteigen. Jede zusätzliche Nutzung eines bereits benutzen Services belastet die Schnittstelle zu dem Service und erhöht die Komplexität des Gesamtsystems [CA88].

Das was das heutige Service Angebot ausmacht ist dass die Services über das Web erreichbar sind. Die Tatsache, dass sie eine allgemeingültige, neutrale Schnittstellensprache haben, nämlich WSDL, macht sie von der Sprache der Benutzerkomponente unabhängig. Es spielt keine Rolle mehr in welcher Sprache die Services verfasst sind. Sie können in jeder beliebigen Sprache geschrieben werden, sie müssen nur die neutrale Schnittstellensprache bedienen können. Das macht es möglich Programmiersprachbarriere zu überwinden. Moderne Service-orientierter Systeme sind also mehrsprachlich. Das unterscheidet sie von konventionellen IT-Systemen, die in der Regel mit einer Sprache implementiert wurden.

Ein weiterer Unterschied ist die Verfügbarkeit der Services. In einem klassischen Client/Server System gehören die Server-Komponenten zum Anwender. Er kann darüber bestimmen was mit ihnen geschieht. Er trägt auch die Verantwortung für deren Wartung und Weiterentwicklung. Web Services sind hingegen irgendwo im Cyberspace. Sie könnten im Besitz des Anwenders sein, müssen es aber nicht. In vielen Fällen werden sie von dem einzelnen Benutzer nur benutzt. Falls mehrere Web-Services zum gleichen Zweck angeboten werden, können sie nach Bedarf ausgetauscht werden. Der Vorteil für den Benutzer ist dass er sie nicht entwickeln und pflegen muss. Der Nachteil ist, dass er sie nicht über sie frei verfügen kann. Sie haben einen eigenen Lebenszyklus, der nicht mit dem der benutzenden Systeme übereinstimmt. Das Web-Service, das der Benutzer heute beansprucht muss nicht mit dem gleichen Service, den er gestern benutzt hat, identisch sein. Er könnte sich geringfügig anders verhalten und andere Ergebnisse liefern. Eventuell wurde er über Nacht durch eine neue Version ersetzt. Wenn der Benutzer kein Service Level Agreement hat muss der Service nicht einmal die gleiche Funktionalität bringen, geschweige denn zur gleichen Qualität. Die Verantwortung für die Anpassung an der neuen Schnittstelle liegt beim Benutzer [TGH04].

Die Tatsache, dass die benutzten Services nicht im Besitz des Benutzers sind hat vielleicht den größten Einfluss auf die Evolution Service-orientierter Systeme, sogar mehr als den der Mehrsprachigkeit. Bisher waren die Dienstprogramme, bzw. die allgemeingültigen Subroutinen und die Klassenbibliotheken, im Besitz des jeweiligen Anwenders. Entweder hat er sie gekauft oder, wie im Falle des Open Source, geschenkt bekommen. Jedenfalls konnte er mit ihnen machen was er wollte. Somit konnte er sie als fertige Komponente in seine Anwendungssoftware einbauen. Sie wurden einen festen Bestandteil davon und konnten zusammen mit den eigenentwickelten Bestandteilen fortgeschrieben werden. Oder sie blieben unverändert sowie der Benutzer sie bekommen hat. Er musste nicht fürchten, dass sie sich über Nacht verändern.

Falls der Benutzer darauf beharrt die Service-Komponente selber zu besitzen, wird es keinen wesentlichen Unterschied zu den klassischen Client/Server Systemen geben, es sei denn das er jetzt das Internet benutzt um auf sie zuzugreifen. Statt über eine Client/Server Schnittstelle wie CORBA-IDL werden sie über eine Webschnittstelle angestoßen. Was die Evolution der Systeme anbetrifft bleibt alles gleich. Der Benutzer muss dafür sorgen, dass die Evolution der Services mit der Evolution der Klienten synchronisiert ist.

Wenn also von Service-orientierter Systeme die Rede ist, müssen wir unterscheiden zwischen Service-orientierten Systemen die im vollen Besitz des Benutzers sind und Service-orientierten Systemen bei denen die benutzten Services nicht der Kontrolle des Benutzers unterliegen, z.B. Software on Demand [Sn07]. Im letzteren Fall stellen sich neue Herausforderungen für die Evolution der Anwendungssysteme heraus, Herausforderungen jenseits der Verteilung und der Mehrsprachigkeit. Alles hat seinen Preis. Der Preis für die Nutzung von Services, die nicht im eigenen Besitz sind, ist das Abhängigkeitsverhältnis zu den Providern jener Services. Zum Einen wird die Evolution, sprich Weiterentwicklung, der eigenen Software von deren Evolution bedingt und zum Anderen wird der Benutzer gezwungen die Evolution der benutzten Services nachzuvollziehen. Kurzum, der Benutzer ist nicht mehr autark [NV00].

Zusammenfassend ist hier festzustellen, dass der Übergang zu service-orientierten Systemen die IT-Benutzer einerseits von der lästigen Pflege und Weiterentwicklung sämtlicher Software-Bausteine befreit, sie jedoch anderseits mit neuen übergeordneten Problemen belastet. Die drei Hauptprobleme sind:

- Die Mehrsprachigkeit,
- Die Schnittstellenkomplexität und
- Die Lieferantenabhängigkeit

Wer Service-orientierte Systeme einsetzen möchte, muss diese drei Probleme im Griff bekommen. Der Rest dieses Beitrages befasst sich mit möglichen Lösungen zu diesen Problemen.

3. Das Problem der Mehrsprachigkeit

In früheren IT-Systemen hatten Anwender in der Regel mit nur einer Programmiersprache zu tun. Alle Entwickler haben diese eine Sprache mehr oder weniger beherrscht. Es gab COBOL Betriebe, C Betriebe oder Java Betriebe. Hinzu kamen allenfalls eine Jobsteuerungssprache, eine Datenbankzugriffssprache wie DLI, ADABAS oder SQL und eine Benutzeroberflächensprache wie BMS, MFS oder HTML. In einem service-orientierten System werden viele Sprachen und damit vielen Technologien zusammenkommen. Die Systeme werden multi-kulturell.

Für die Benutzeroberflächen kommen außer HTML auch XHTML, XSL und neuere Internetsprachen wie PHP und Flex ins Spiel. Für die Systemschnittstellen können neben IDL, XML und WSDL verwendet werden. WSDL ist die Sprache für die Anbindung der Web Services. Die Web Services selbst können in jeder beliebigen Sprache verfasst werden – PL/I, COBOL, C++, C#, Java, ADA, PERL, Natural, ABAP, usw. Schließlich gibt es neben der traditionellen Clientsprachen wie C++, C#, Java und PHP die neuen Geschäftsprozesssprachen wie BPEL, BPMN und JCom sowie weitere Domain Specific Sprachen. Der Turm von Babel lässt grüßen [Jon03].

Ein IT-Betrieb der die Sourcen seiner Service-orientierten Systeme zwecks der Messung oder der Nachdokumentation analysieren möchte muss Werkzeuge einsetzen, die alle seiner benutzten Sprachen abdecken. Denn es bringt wenig allein nur die Clientseite zu analysieren ohne die Services und deren Schnittstellen einzubeziehen. Man will schließlich wissen wie groß und wie komplex das ganze System ist oder wie ist das ganze System zusammengesetzt, und das kann man nur herausbekommen wenn man alle Bestandteile analysiert.

Eine weitere Folge der Mehrsprachigkeit ist der erhöhte Personalbedarf. Ein einzelner Mitarbeiter wird kaum mehr als zwei Sprachen beherrschen. Viele werden nur eine einzige Sprache beherrschen. Dennoch braucht der Systembetreiber für die Evolution des Systems Spezialisten für jede Sprache damit er jede Sprache fortschreiben kann. Auch wenn die einzelnen Spezialisten meistens nicht ausgelastet sind, werden sie für alle Fälle gebraucht. D.h. der Anwender braucht nicht weniger sondern noch mehr Wartungspersonal als bisher der Fall war. Die Erhaltungskosten Service-orientierter Systeme steigen.

Eine besondere Herausforderung stellt die Pflege und Weiterentwicklung von Legacy Softwarebausteine, die als Web Services gekapselt und angeboten werden. Dies wird öfters notwendig sein wenn ein Standard Service nicht verfügbar und die Entwicklung einer neuen Service zu teuer ist. In dem Fall muss der Anwender auf seine bestehende Software zurückgreifen und sie web-fähig gestalten. Die Kapselungstechnologie ist eine ausgereifte und weit verbreitete Methode zur Wiederverwendung bestehender Softwarekomponente als Services [Sn06]. Sie ist zudem schnell und preiswert, vor allem wenn die Kapselung automatisch stattfindet. Sie hat aber einen Haken. Der alte Code bleibt und lebt weiter als Bestandteil, bzw. als Organtransplant, des neuen service-orientierten Systems. Als solcher muss er korrigiert, adaptiert und möglicherweise erweitert werden. Der Anwender braucht also Personal, das mit dem alten Code umgehen kann. Auch wenn diese eingebauten Altbausteine sich nur geringfügig ändern, braucht man trotzdem Personal um sie zu betreuen. Ihr Know-How ist für die Erhaltung dieser Services, sprich für die Kontinuität der Dienstleistung, unentbehrlich.

Die Lösung zu diesem Personalproblem liegt in der Konzentration der Ressourcen. Nur große Organisationen können es sich leisten so viele teure Spezialisten parat zu halten. Die Anwender werden gezwungen ihre Personalressourcen in größere IT-Service Zentren zusammen zu fassen. Kleine und mittelgroße Anwender, die es sich nicht leisten können so viel Know-How anzueignen und zu halten, müssen zumindest Teile der Evolutionsarbeiten outsourcen. Entweder benutzen sie fertige Services on Demand oder sie übergeben ihre maßgeschneiderten Services an einen externen Partner, der sie für sie wartet. Sie werden selber nur Teile ihrer Gesamtsysteme, wo möglichst die Clientkomponente selber pflegen können. Der Preis der anspruchsvollen und diversifizierten Service-Technologie ist die Aufgabe der Selbstständigkeit. Nur die größten Anwender werden in der Lage sein alles selber zu machen. Dem Rest wird nicht anders übrig bleiben, wenn sie mitziehen wollen, als in die Abhängigkeit externer Dienstleister zu geraten.

4. Das Problem der steigenden Komplexität

Auch die großen IT-Anwenderbetriebe werden sich mit der zunehmenden Komplexität der Service-orientierten Systeme schwer tun. Sämtliche Evolutionsaufgaben, sei es Fehlerkorrekturen, Änderungen oder der Einbau neuer Funktionalität müssen über Teilsystemgrenzen und Schnittstellen hinweg miteinander abgestimmt werden. Sonst besteht die Gefahr dass alles auseinander läuft. Zu viel sind die gegenseitigen Abhängigkeiten zwischen den verteilten Services auf der einen Seite und den verteilten Clientprozessen auf der Anderen.

Um überhaupt Herr dieser komplexen Lage zu werden braucht die zuständige Entwicklungsorganisation eine detaillierte, graphische Landkarte der IT Landschaft [BSB08]. Aus dieser Karte geht hervor welche Prozesse welche Services verwenden, welche Services welche anderen Services in Anspruch nehmen und welche Prozesse mit einander direkt verbunden sind. Dazu muss man sehen können, auf welche Datenbanken die Services zu greifen und wie diese Datenbanken zusammenhängen. Schließlich sollten alle Schnittstellen zu fremden Systeme erkennbar sein. Diese graphische Landkarte ist die sichtbare Oberfläche zu einer darunter liegenden Repository, die automatisch aus der Analyse etlicher Sourcen, Modelle und Dokumente gefüttert wird. Jedes Mal wenn ein neues Release der Software zusammengestellt wird, ist die Repository über die statische Analyse der geänderten Komponente auf den neuesten Stand zu bringen [SHT04].

Sämtliche Fehlermeldungen, Änderungsanträge und neue Anforderungen müssen zunächst auf der großen Landkarte des betroffenen Systems fixiert werden. Innerhalb der Repository werden sie dann durch das System vom Baustein zu Baustein verfolgt und alle betroffene Knoten im Netzwerk der Prozesse, Klienten und Services identifiziert bis der Auswirkungsbereich der Änderung genau abgesteckt ist. Die Größen der betroffenen Bausteine werden aus der Metrikdatenbank geholt und summiert um die Summe dann durch die Änderungsrate zu glätten [Sn01]. Dadurch lässt sich der Umfang der geplanten Änderung herausbekommen. Dies sei erforderlich um die Kosten zu schätzen. Diese Kosten werden mit dem verrechneten Nutzen verglichen um zu sehen ob die Änderung sich lohnt. Nur lohnende Änderungen sind in solchen komplexen Systemen erlaubt, denn in einem Service-orientierten System sind die Risiken einer Änderung viel zu hoch [Sn10].

Wenn einmal entschieden ist das System zu ändern, muss der zu ändernden Bausteine bezüglich deren Abhängigkeiten genau untersucht werden, um ja keine unerwünschte Nebeneffekte zu verursachen. Änderungen zu Schnittstellen sind besonders heikel. Es muss vorher geklärt werden, welche Services welche Schnittstellen benutzen. Dies ist wohl nur mit einer Repository möglich. Deshalb ist ein mächtiges Repository eine unabdingbare Voraussetzung für die Evolution eines Service-orientierten Systems.

Nach der Vollendung der Änderungen sind zunächst die geänderten Softwarebausteine in einer künstlichen Umgebung neu zu testen. Ihre Verknüpfungen nach Außen werden dabei durch Stubs simuliert. Erst wenn jeder Netzknote für sich getestet worden ist, werden die Knoten verbunden und in ihrer Gesamtheit getestet. Es wäre zu aufwendig werden jede Interaktion zu den Services einzeln zu testen. Dies wird nur gemacht wenn ein Problem auftritt, z.B. wenn sich der Auftrag von einem Prozess-Schritt an ein Service nicht korrekt abgehandelt wird. Der Integrationstest beschränkt also auf den Test einzelner Interaktionen. Ansonsten ist das Ganze als System zu testen.

In der Evolution wird mit den angereicherten Daten der vorhergehenden Version getestet. Die Daten werden lediglich entsprechend den Änderungen mutiert. D.h die Tester müssen wissen welche Auswirkung die Änderungen auf ihre Testdaten haben werden. Um dies zu erfahren brauchen auch sie Zugang zum System-Repository. Über das Repository werden sie erfahren welche Testfälle sie fahren müssen um die Zielschnittstellen zu treffen. Es wird allzu leicht sich im Test zu verzetteln wenn blind getestet wird. Es komme darauf an gezielt zu testen und dies sei nur möglich mit Hilfe geeigneter Analysewerkzeuge [NJH03].

Zusammenfassend ist festzustellen, dass die Komplexität Service-orientierter Systeme nur mit leistungsfähigen Werkzeugen zu bewältigen ist. Diese werden reichen von einfachen Analyse Tools bin hin zum einem allumfassenden Repository und zum vollautomatisierten Regressionstest. Wer also Service-orientierter Systeme einführen will, muss bereit sein in eine voll ausgebaute Werkzeuginfrastruktur zu investieren.

5. Das Problem der Lieferantenabhängigkeit

In der IT-Welt mit ihren vielen Spezialgebieten bei denen jedes ein eigenes Teilwissen voraussetzt, hat es schon immer Abhängigkeiten gegeben. Die Analytiker sind von den Programmierern abhängig, da sie den Code beherrschen. Die Programmierer sind von Datenbankadministratoren abhängig weil diese über die Daten verfügen, auf die die Programmierer zugreifen müssen. Alle drei sind von den Systemadministratoren abhängig, weil sie die Herren jener Ressourcen sind welche die Anderen zu ihrer Arbeit brauchen. Wir sind in der IT-Welt gewohnt von Anderen abhängig zu sein. Diese Abhängigkeiten waren aber alle innerhalb der Anwenderorganisation und konnten intern geregelt werden. Am Ende des Tages hatten alle den gleichen Chef.

Mit der Einführung Service-orientierter Systeme treten neue Abhängigkeiten auf. Wer fremde Services in seine Geschäftsprozesse einbaut ist von den Lieferanten jener Services abhängig. Möglicherweise ist dieser eine interne Dienststelle, aber oft wird es sich um einen externen Lieferanten handeln. Wie früher mit den mathematischen Subroutinen, werden viele Services besondere Kenntnisse voraussetzen, die nur wenige Service Provider haben. Das ist letztendlich der Sinn von Services. Der Benutzer holt sich verpacktes Wissen nach Bedarf, Wissen das er im eigenen Unternehmen nicht zur Verfügung hat. Daraus entsteht ein Abhängigkeitsverhältnis. Darum ist es wichtig mit jedem externen Service Lieferant ein Service Level Agreement – SLA – abzuschließen [Sn05]. Auch mit internen Dienststellen die Services bereitstellen, empfiehlt es sich ein SLA zu haben. Die Service Benutzer müssen dafür Sorge tragen, dass die sie ins Leben rufen weiterentwicklungsfähig sind. Diese können jedoch nur so bleiben wenn alle ihrer Bestandteile ebenfalls weiterentwickelt werden. Jede Service-Komponente wird ein eigener Lebenszyklus haben. Die Wenigsten werden so bleiben wie sie sind, die Meisten werden regelmäßig erneuert. Über das Service Level Agreement kann sich der Benutzer eines Services sichern, dass die Evolution des Services mit der Evolution seiner eigenen Software im Einklang bleibt [Bö08].

In der SLA gilt es zu regeln wie oft die Services erneuert werden und wie der Benutzer darüber zu informieren ist. Es darf nicht vorkommen, dass ein Service ohne adäquate Vorwarnung geändert wird. Der Benutzer braucht genügend Zeit seine eigene Software auf das veränderte Service einzustellen und genügend Zeit die Interaktion mit dem veränderten Service zu testen. Das Testen geänderter, bzw. erneuerter, Services wird die meiste Zeit in Anspruch nehmen. Um diese Zeit möglichst kurz zu halten, braucht der Benutzer genaue Angaben darüber was sich geändert hat. Zur Sicherung der Änderungen bedarf es eine besondere Schnittstellenverwaltung. Je mehr unter unterschiedliche Services beansprucht werden, desto aufwendiger diese Verwaltung.

Das Service Level Agreement sollte sowohl die Funktionalität als auch die Qualität des Services regeln. Was die Funktionalität anbetrifft sind:

- Die einzelnen Operationen
- die Eingangsparameter und
- die zu erwarteten Ergebnisse

zu spezifizieren. Bezüglich der Qualität sind u.a.

- die Zuverlässigkeit
- die Verfügbarkeit
- die Sicherheit
- die Antwortzeit und
- die statische Qualität der Schnittstelle

zu regeln.

Die Zuverlässigkeit wird im Bezug zur Anzahl Abstürzen und fehlerhafte Ergebnisse pro 1000 Aufträge definiert. Die Verfügbarkeit wird im Prozentsatz der Betriebszeit definiert. Die Sicherheit wird anhand der vereinbarten Sicherheitsmaßnahmen vorgegeben. Die Antwortzeiten werden in Sekunden zwischen dem Absenden eines Auftrages und dem Erhalt der Antwort festgelegt. Die statische Qualität der Service-Schnittstelle wird über eine Reihe Regel und Metrik zur Gestaltung der Schnittstellen vereinbart [Sn07].

Es wird aber nicht genügen eine SLA auf Papier zu haben. Die Einhaltung der SLA muss überwacht werden. Die geringste Abweichung von der Vereinbarung ist sofort zu registrieren und zu melden. Daraufhin tritt der vorbestimmte Eskalationsprozess in Gang. Auch dieser Prozess muss in der SLA beschrieben sein. Kurzum das Service Level Agreement spielt eine zentrale Rolle bei der Evolution eines Service-orientierter Systems. Folglich wird eine der Haupttätigkeiten eines Projektleiters sein, solche SLAs zu verfassen und mit den Service Providern auszuhandeln. Anschließend ist er verpflichtet die Umsetzung der SLAs zu verfolgen. Dazu braucht er einen Produktleitstand mit dessen Hilfe er die Einhaltung der einzelnen Services kontrollieren kann. Der Produktleitstand soll an dem Repository angeschlossen sein, und über alle Informationen zu dem Produkt verfügen. Über die Key Produktindikatoren (KPIs), wie z. B., Antwortzeitverhalten und Fehlerrate, sowie die Prozessindikatoren, wie z.B., Zeit zur Fehlerkorrektur und Zeit bis zur Änderungsumsetzung, wird er in der Lage sein den Evolutionsprozess zu steuern.

6. Schlußfolgerung

In diesem Beitrag wurde angesprochen, welche Probleme in Zusammenhang mit der Evolution Service-orientierter Systeme zu erwarten und wie sie zu begegnen sind. Diese Probleme werden erst nach der Einführung der Systeme richtig in Erscheinung treten. Die erhöhte Komplexität verteilter und vernetzter Services mit deren vielen Schnittstellen zueinander wird die Fortschreibung solcher Systeme erschweren und die Risiken jeder Änderung steigern. Die Mehrsprachigkeit der beteiligten Servicekomponente wird die Kosten der Systemevolution weiter in die Höhe treiben. Der Betreiber der Systeme muss für jede verwendete Sprache Personal halten, das sich mit der jeweiligen Sprache auskennt. Hinzu kommt die zunehmende Abhängigkeit von den Service-Lieferanten. Der Benutzer muss sich gegen unerwartete Funktionsänderung und Service-Leistungsverfall schützen. Das Instrument dazu ist das Service Level Agreement. Damit steigt aber der Verwaltungsaufwand. In jedem Projekt muss es jemanden geben, der sich um die SLAs kümmert.

Literaturverzeichnis

[BL75] Belady, L. /Lehman, M.: „The Evolution Dynamics of Large Programs“, IBM Systems Journal, Nr. 3, Sept. 1975, S. 11

[LB85] Lehman, M., Belady, B.: “Program Evolution, Academic Press, London, 1985, S. 29

Seacord, R./Plakosh, D./Lewis, G.: Modernizing Legacy Systems, Addison-Wesley, Boston, 2003

[Sn02] Sneed, H: "Das Ende von Migration und Reengineering", Computerwoche Nr. 7, Feb. 2002, S. 18

[Sn04] Sneed, H.: "A Cost Model for Software Maintenance and Evolution", in Proc. of 20th ICSM, IEEE Press, Chicago, 2004, S. 264

[vH05] von Hahn:, E.: Werterhaltung von Software – Planung und Bewertung von Reengineering Maßnahmen am Beispiel von Standard Software, Deutscher Universitätsverlag, Wiesbaden, 2005, S. 173

[BR00] Bennett, K. H.; Rajlich, V. T.: Software Maintenance and Evolution: A Roadmap. In Finkelstein, A. [Hrsg.]: The Future of Software Engineering, Proceedings of the International Conference on Software Engineering, ACM Press, 2000, S. 73

[KBS05] Krafzig, D./Banke, K./Slama, D.: Enterprise SOA – Service-Oriented Architecture, The Coad Series, Prentice-Hall, Upper Saddle River, N.J. 2005, S. 9

[SL07] Smith, D.; Lewis, G.: "Standards for Service-oriented Systems", in Proc. of 11th European Conference on Software Maintenance and Reengineering, Amsterdam, 2007, S. 100

[NL05] Newcomer, E./Lomow, G.: Understanding SOA with Web Services, Addison-Wesley, Upper Saddle River, N.J., 2005, S. 10

[Lo93] Lorenz, M.: Object-oriented Software Development, Prentice-Hall, Englewood Cliffs, 1993, S. 125

[Mo91] Montgomery, S.: AD/Cycle – IBM's Framework for Application Development and CASE, Van Nostrand, New York, 1991, S. 17

[SC99] Sadiq, W., Cummins, F.: Developing Business Systems with CORBA, Cambridge University Press, Cambridge, 1999, S. 23-42

[Ke02] Keller, W.: Enterprise Application Integration – Erfahrungen aus der Praxis, dpunkt Verlag, Heidelberg, 2002, S. 5

[CHK06] Conrad, S./ Hasselbring, W./ Koschel, A./ Tritsch, R.: Enterprise Applcation Integration, Elsevier Spektrum Verlag, Heidelberg, 2006, S. 11

[SnSn03]Sneed, H./ Sneed, S.: Web-basierte Systemintegration, Vieweg Verlag, Wiesbaden, 2003, S. 91

[CA88] Card, D., Agresti, W.: "Measuring Software Design Complexity", Journal of Systems & Software, Vol. 8, 1988, S. 185

[TGH04] Tilley,S./Gerdes,J./Hamilton,T./Huang,S./Mueller,H./Smith,D./Wong,K.: "On the Business Value and technical Challenge of adopting web services", Journal of Software Maintenance and Evolution, Vol. 16, Nr. 1-2, Jan. 2004, S. 31

[Sn07] Sneed, H.: "Migrating to Web Services – A Research framework", in Proc. Of 11th CSMR, Amsterdam, March, 2007, S. 116

[ABC01] Abraham, N.; Bibel, U.; Corleone, P.: Formatting Contributions for LNI. In (Glück, H.I. Hrsg.): Proc. 7th Int. Conf. on Formatting of Workshop-Proceedings, New York 1999. Noah & Sons, San Francisco, 2001; S. 46-53.

[Az99] Azubi, L. et.al.: Die Fußnote in LNI-Bänden. In (Glück, H.I.; Gans, G., Hrsg.): Formattierung leicht gemacht – eine Einführung. Format-Verlag, Bonn, 1999; S. 135-162

[Ez99] Ezgarani, O.: The Magic Format – Your Way to Pretty Books, Noah & Sons, 2000.

Towards the Combination of Clustering-based and Pattern-based Reverse Engineering Approaches

Oleg Travkin, Markus von Detten, Steffen Becker
Software Engineering Group, Heinz Nixdorf Institute
Department of Computer Science, University of Paderborn, Paderborn, Germany
[oleg82|mvdetten|steffen.becker]@mail.uni-paderborn.de

Abstract: Reverse Engineering, i.e. the analysis of software for the purpose of recovering its design documentation, e.g. in form of the conceptual architecture, is an important area of software engineering. Today, two prevalent reverse engineering approaches have emerged: (1) the clustering-based approach which tries to analyze a given software system by grouping its elements based on metric values to provide the reverse engineer with an overview of the system and (2) the pattern-based approach which tries to detect predefined structural patterns in the software which can give insight about the original developers' intentions. These approaches operate on different levels of abstraction and have specific strengths and weaknesses. In this paper, we sketch an approach towards combining these techniques which can remove some of the specific shortcomings.

1 Introduction

During their life cycle, software systems have to be maintained and continuously extended. In order to perform these tasks, a developer has to first understand the design of the software which can be a tedious and complicated task due to incomplete design documentation. Reverse engineering is the task of analyzing a software system in order to recover its design documentation. In the past, two types of approaches for the reverse engineering of software systems have been proposed: clustering-based reverse engineering and pattern-based reverse engineering [Sar03].

Clustering-based reverse engineering approaches try to identify the system's architecture by grouping its basic elements (e.g. classes) based on the values of certain metrics. Examples of common metrics used for this purpose are coupling and stability [Mar94]. The result of a clustering-based analysis is a set of packages or components (plus optionally their dependencies or connectors) which represents a possible system architecture. Because the calculation of metric values is relatively easy for most metrics, a clustering-based analysis can be carried out in short time frames, even for large systems. A major drawback of this approach, however, is the high level of abstraction that comes with the use of metrics. Some complex architecture-relevant information cannot easily be captured by metric formulas. In addition, considering more details also complicates the calculation which would greatly increase the time consumption of the approach. An overview of clustering-based reverse engineering approaches is given for example by Ducasse and Pollet [DP09].

Pattern-based reverse engineering approaches are concerned with the detection of predefined patterns in the software. In this paper, we only consider structural patterns. Patterns exist for different levels of abstraction, different domains and purposes but the most famous collection of patterns are the *Design Patterns* by Gamma et al. [GHJV95]. The rationale behind their detection is that each design pattern has a specific intent that describes in which situation the pattern should be applied. Hence, the detection of a design pattern can reveal the developer's design intentions and thereby foster the understanding of the software. Another renowned category of patterns are the Anti Patterns [BMMM98] which describe frequently occurring bad solutions to common problems together with refactorings to correct them. The detection of patterns in general is the subject of many scientific approaches of which Dong et al. give an overview [DZP09]. Because pattern detection often takes much more detailed information into account than clustering-based approaches, it is generally much slower. Moreover, due to their low level of abstraction, an impractically large set of pattern implementations can be detected even for medium-scale systems.

In order to combine the strengths and overcome the limitations of both approaches, we propose a reverse engineering process which first performs a metric-based clustering for a given system to identify components and afterwards applies a pattern detection mechanism to each of the identified components.This combination mainly addresses two problems. It improves the pattern-based analysis by providing a more focused approach than a complete analysis of the whole system and thereby reduces the required time and the number of detection results. Second, the higher level of detail that is taken into account during a pattern-based analysis can be utilized to refine the architecture recovered by the clustering-based analysis: The pattern detection can identify certain patterns which have an influence on the metrics used by the clustering and which can therefore affect the recovered architecture. An example of this is presented in Section 3.

We intend to build a prototype reverse engineering tool by combining the Software Model Extractor SoMoX developed at the FZI Karlsruhe [BHT+10] and the pattern detection tool suite Reclipse developed at the University of Paderborn [vDMT10]. While SoMoX uses a weighted combination of metrics to infer the architecture of industry size systems, Reclipse is capable of detecting arbitrary structural patterns whether they are design or anti patterns.

The contribution of this paper is a reverse engineering approach that combines clustering-based and pattern-based analyses to improve the reverse engineered results and ultimately provide more accurate architectural information. The following section briefly presents the proposed combined detection process. Section 3 shows an example which benefits from this combination. In Section 4 we draw conclusions and sketch future work.

2 The combined reverse engineering process

Figure 1 presents the main steps of the combined reverse engineering process and the different artifacts involved in it. The two inputs of the clustering step are the source code of the system and a set of metrics. The result is the heuristically recovered architecture

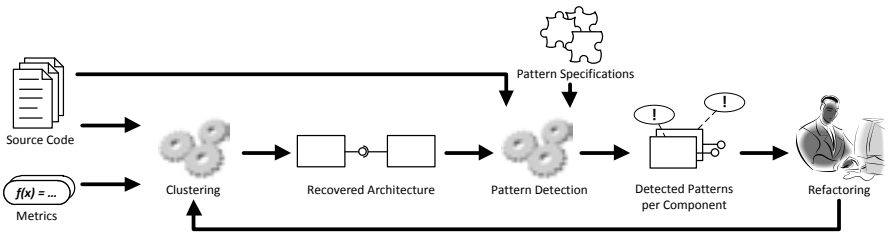


Figure 1: Illustration of the combined reverse engineering process

of the software. The source code together with the recovered architecture and a set of structural pattern specifications then serves as an input for the pattern detection algorithm. Because the pattern detection is scoped on the detected components, the required analysis time and the number of results are obviously reduced. On the other hand, patterns that exist across component boundaries are neglected by the scoped approach. This problem is subject of future work. The detected patterns can serve as a starting point for refactorings which can have an impact on the architecture of the software. Hence, a clustering of the refactored system may lead to a different and possibly improved recovered architecture.

3 Possible influences of anti patterns on the conceptual architecture

One of our basic assumptions is that that the existence of certain patterns can have an influence on the architecture recovered by the clustering. For the sake of brevity, we restrict our discussion in this paper to anti patterns.

As the clustering relies on code metrics like coupling and stability, anti patterns which affect these metrics can adulterate the clustering results. One example for this is an illegal direct access, an anti pattern which increases the coupling between two classes. This is illustrated in the following example.

The upper left part of Figure 2 shows exemplary source code for a simple system with two interfaces IA and IB and two classes A and B which implement those interfaces. IA declares a method m1 and IB declares a method m2. Class A implements IA and has a reference to IB. Class B, implementing IB, additionally declares a method m3. The implementation of m1 in class A downcasts the object contained in the field ib to the type B and then calls m3, thereby circumventing the use of the interface. Part b) of the figure shows a class diagram representation of the same system which clarifies that A references both, IB and B. The coupling between A and be is therefore 1.0 [1].

The use of the concrete type instead of the interface violates one of the programming principles defined by Gamma et al.: *"Program to an interface, not an implementation."*

[1] The coupling between A and B is computed as the ratio between the accesses from A to B and the number of accesses from A to all classes if the system. In this simple, illustrative example $Coupling(A, B) = \frac{1}{1} = 1$.

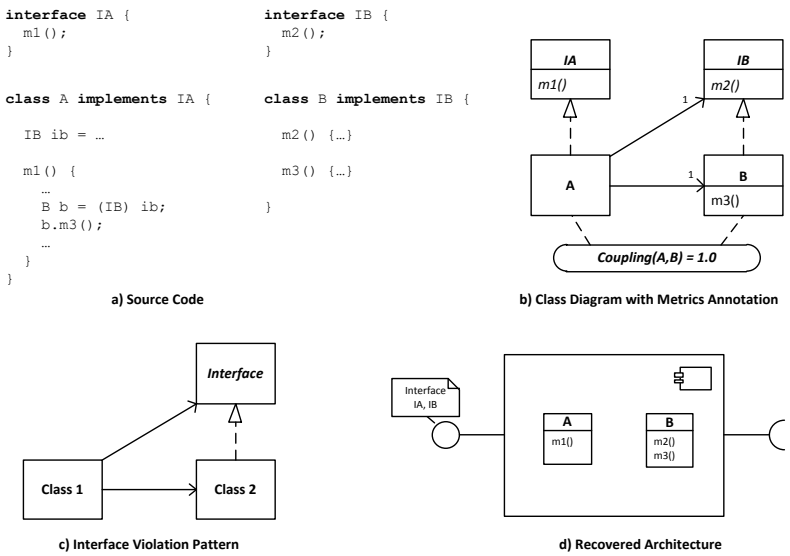


Figure 2: Violation of an interface and the resulting component

[GHJV95]. It also leads to a tight coupling between classes A and B which should have been prevented by the use of interfaces. This is a design flaw known as *Interface Violation* which is depicted in the lower left part of Figure 2. The existence of this flaw can have several reasons: for example the declaration of m3 could simply be missing in IB or m3 may not supposed to be accessed from the outside. Either way, the high value of the coupling metric would probably cause clustering-based architecture recovery tools to group A and B together, for example as members of the same software component as shown in part d) of Figure 2.

Figure 3 shows a refactored version of the code from Figure 2. Method m3 has been added to the interface IB so that the downcast in class A becomes unnecessary. The two classes now communicate directly via their implemented interfaces which reduces their direct coupling to 0. The class diagram representation of the refactored code clarifies that A no longer has a direct reference to B.

The refactoring removes the interface violation from the system. Because this design change affects the coupling of A and B, it can have an impact on the clustering of the system. An architecture recovery tool might now group the classes as depicted in the lower part of Figure 3. A and B would be assigned to two different software components which communicate solely via their interfaces, i.e. the component containing class B provides the interface IB which is required by the component containing A.

The detection and removal of the interface violation consequently leads to a change in the reverse engineered architecture. We were able to detect several of the described interface violations in the reference implementation of the Common Component Modeling Example

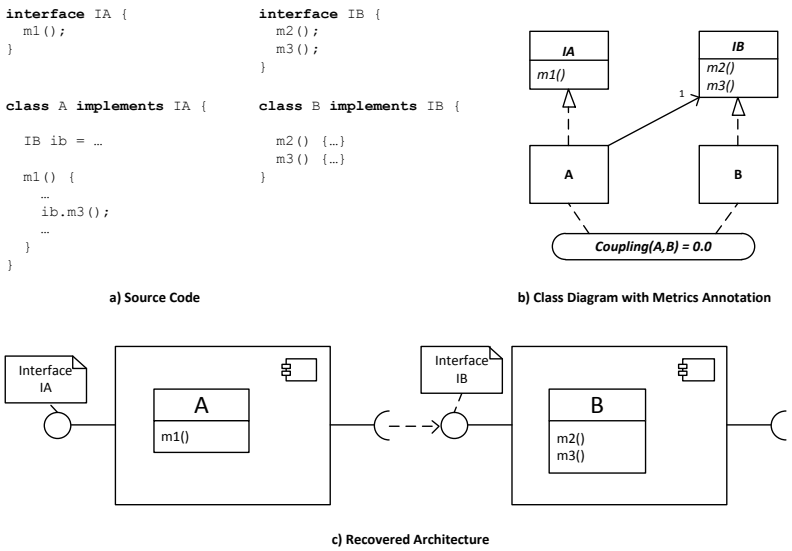


Figure 3: Refactored code with the improved component structure

CoCoME [HKW+08] which could prove to have significant impact on the clustering of the implementation with current reverse engineering tools (e.g. SoMoX [BHT+10]).

4 Conclusions and Future Work

We sketched an approach which combines clustering-based and pattern-based reverse engineering techniques and suggested that this combination can have multiple advantages. Pattern-based reverse engineering approaches can be used for the identification of (anti) patterns that affect the clustering. The removal of these patterns can then improve the recovered architecture. Second, we argued that a clustering of the system prior to the detection of patterns could enhance the focus and speed of the detection and would not overwhelm the reverse engineer with too many results.

Our research is still in its early stages and numerous points remain to be considered. First, we want to identify more patterns which can have an influence on the architecture recovery. It must also be noted that not all occurrences of a pattern necessarily have the same impact. Interface violations between two classes are only critical if the two classes conceptually belong to different components. Classes in the same package may as well be expected to be tightly coupled. We suppose that these cases can be distinguished by a relevance analysis step that takes additional metrics into account. In the example, an Interface Violation could only be marked as critical if the two classes reside in different packages and are otherwise coupled to independent sets of other classes.

As mentioned in Section 2, the confinement of the pattern detection to the detected components results in false negatives which has to be considered in future work.

We also plan to extend the reverse engineering process by additional automated steps that help the reverse engineer with further design decisions. For example, when a critical Interface Violation has been detected, it could be removed in several different ways by automatic transformations (which extend the interface, remove the violating access, etc.). These transformations can have different impacts on the clustering which could be analyzed and presented to the reverse engineer to improve his knowledge about the choices at hand.

In addition, it would be interesting to evaluate different implementations of the CoCoME example. We discovered several interface violations in the reference implementation but implementations with component frameworks like SOFA or FRACTAL may have different issues. This could help to understand to which extent the choice of an implementation framework affects the quality of the final implementation.

References

[BHT+10] S. Becker, M. Hauck, M. Trifu, K. Krogmann, and J. Kofron. Reverse Engineering Component Models for Quality Predictions. In *Proc. of the 14th European Conference on Software Maintenance and Reengineering*, pages 199–202. IEEE, 2010.

[BMMM98] W. J. Brown, R. C. Malveau, H. W. McCormick, and T. J. Mombray. *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, 1998.

[DP09] S. Ducasse and D. Pollet. Software Architecture Reconstruction: A Process-Oriented Taxonomy. *IEEE Transactions on Software Engineering*, 35(4):573–591, 2009.

[DZP09] J. Dong, Y. Zhao, and T. Peng. A Review of Design Pattern Mining Techniques. *Int. Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 19(6):823–855, 2009.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[HKW+08] S. Herold, H. Klus, Y. Welsch, C. Deiters, A. Rausch, R. Reussner, K. Krogmann, H. Koziolek, R. Mirandola, B. Hummel, M. Meisinger, and C. Pfaller. CoCoME - The Common Component Modeling Example. In *The Common Component Modeling Example*, volume 5153 of *LNCS*, pages 16–53. Springer, 2008.

[Mar94] R.C. Martin. OO Design Quality Metrics - An Analysis of Dependencies. 1994.

[Sar03] K. Sartipi. Software Architecture Recovery based on Pattern Matching. In *Proc. of the Int. Conference on Software Maintenance*, pages 293–296. IEEE, 2003.

[vDMT10] M. von Detten, M. Meyer, and D. Travkin. Reverse Engineering with the Reclipse Tool Suite. In *Proc. of the 32nd Int. Conference on Software Engineering*, volume 2, pages 299–300. ACM, 2010.

Kompatibilitätsanalyse bei Evolution framework-basierter Anwendungen

Fabian Christ
s-lab - Software Quality Lab
Universität Paderborn
fchrist@s-lab.upb.de

Jan-Christopher Bals
Universität Paderborn
jan-christopher.bals@upb.de

Abstract: Die Entwicklung betrieblicher Informationssysteme basiert auf dem Einsatz von Frameworks. Diese bieten ein hohes Maß an Wiederverwendung und sind flexibel anpassbar. Mit der Evolution der eingesetzten Frameworks unabhängig von der Anwendung entsteht die Notwendigkeit, Frameworks durch neuere Versionen zu ersetzen, um Fehler zu beheben oder neue Funktionen benutzen zu können. Etwaige Inkompatibilitäten neuer Versionen erfordern Anpassungen an der Anwendung. In der Praxis entsteht das Problem, dass die erforderlichen Anpassungen schwierig zu bestimmen sind. In dieser Arbeit zeigen wir einen Ansatz zur automatischen Kompatibilitätsanalyse bei der Evolution framework-basierter Anwendungen.

1 Einführung

Ein Unternehmen, das die Entwicklung eines neuen betrieblichen Informationssystems in Auftrag gibt, erwartet, dass dies zu einem fest kalkulierten Preis und definierten Zieltermin geschehen kann. Damit dies für den Softwarehersteller möglich wird, muss dieser viele Teile des neuen Systems aus bereits existierenden Komponenten aufbauen, um bei der Entwicklung nur noch kundenspezifische Teile ergänzen zu müssen. Zentrales Konzept ist der Einsatz von Frameworks, die als Gerüste bereits Teile des Systems implementieren und nur noch durch eigene Erweiterungen spezialisiert werden müssen.

In der Literatur finden sich eine Reihe von Definitionen für Frameworks [JF88, Deu89, Sch97], wobei wir uns auf eine Definition für "Application Frameworks" von Johnson aus dem Jahre 1997 beziehen wollen [Joh97]. Danach ist ein Framework a) definiert durch seine wiederverwendbare Struktur: "A framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact." und b) durch seinen Zweck als domänenspezifisches anpassbares Gerüst einer Anwendung: "A framework is the skeleton of an application that can be customized by an application developer.". Die Anpassung erfolgt dabei an so genannten Erweiterungspunkten, für die entsprechende Erweiterungen entwickelt werden können, die das Verhalten der Anwendung steuern. Aufgrund des hohen Grads an Wiederverwendungsmöglichkeiten sind Frameworks zentraler Bestandteil heutiger Softwaresysteme.

Wir betrachten in Abbildung 1 das Beispiel einer Anwendung, das ein Framework für die Programmierung der grafischen Benutzeroberfläche (GUI) verwendet. Das Framework

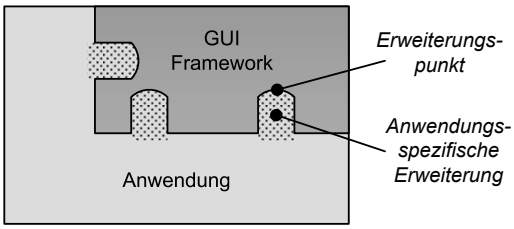


Abbildung 1: Anwendung verwendet ein GUI-Framework

bietet die Möglichkeit vorgefertigte Steuerelemente wie Dialoge und Listen zu verwenden. Mit Hilfe entsprechender *Erweiterungspunkte* des Frameworks können eigene neue Steuerelemente implementiert werden.

Nachdem eine Anwendung mit Hilfe des Frameworks entwickelt und ausgeliefert wurde, betritt sie die Weiterentwicklungsphase im Entwicklungszyklus. Jedoch nicht nur die Anwendung im Sinne anwendungsspezifischer Erweiterungen wird weiterentwickelt, sondern auch das eingesetzte Framework evolviert unabhängig davon. Unterschiedliche Entwicklungs- und Releasezyklen führen dazu, dass sich die beteiligten Teile unterschiedlich schnell weiterentwickeln. Es kommt zu der Situation, dass für geplante Funktionen der Anwendung das Framework durch eine neue Version aktualisiert werden muss. Jedoch stellt sich die Frage, welche Auswirkungen eine solche Migration der Anwendung hat.

In Abbildung 2 betrachten wir das Problem einer Anwendung in Version 2.3.0, die ein GUI-Framework in Version 1.1.5 verwendet. Nun soll die Anwendung zu Version 2.4.0 weiterentwickelt werden und dabei ggf. auch das GUI-Framework von Version 1.1.5 auf Version 1.2.0 aktualisiert werden. Ob während der Weiterentwicklung der Anwendung von Version 2.3.0 auf 2.4.0 das GUI-Framework aktualisiert wird, hängt davon ab, ob die neue Frameworkversion ebenfalls kompatibel zur bestehenden Anwendung ist. Je nachdem wie viel Aufwand es bedeuten würde, entstehende Inkompatibilitäten zu beheben, wird man sich unter Zeit- und Kostendruck für oder gegen eine Migration entscheiden.

Die grundlegende Frage im dargestellten Problem bezieht sich darauf, ob die bestehende Anwendung kompatibel zur neuen Frameworkversion ist. Wir wollen daher die ge-

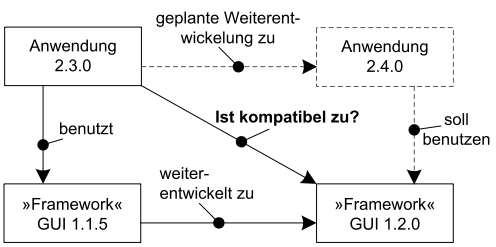


Abbildung 2: Problemstellung

plante Weiterentwicklung der Anwendung außer Acht lassen und das Problem darauf einschränken, dass die Anwendung als unverändert betrachtet wird und das Framework aktualisiert werden muss.

Um die Kompatibilität zu beurteilen, sucht man nach auswertbaren Informationen und stößt dabei meist schnell an Grenzen, da in der Regel der Fall einer speziellen Migration nicht dokumentiert ist. Häufig finden sich gar keine Informationen, die eine verlässliche Beurteilung der Situation ermöglichen würde. Aus diesen Gründen ist die industrielle Praxis in einer solchen Situation, dass man einen oder mehrere Entwickler abstellt, um die Migration probehalber durchzuführen. Dies bedeutet, dass die Entwickler in einem manuellen Prozess versuchen die Anwendung mit dem neuen Framework zu übersetzen und zu starten. Auftretende Fehlermeldungen werden manuell analysiert, im Internet recherchiert und mögliche Lösungen gesucht. Ziel ist es, wieder eine lauffähige Anwendung zu erhalten, so dass man anschließend eventuell vorhandene funktionale Tests durchführen kann. Dieses Vorgehen hat eine Reihe von Nachteilen:

- Es ist ein manueller und damit zeit- sowie kostenaufwendiger Prozess.
- Entwickler werden gebunden, die ansonsten für andere (wichtigere) Entwicklungen eingesetzt werden könnten.
- Der Aufwand einer Migration auf Probe ist im Worst-Case genauso hoch wie die einer tatsächlichen Migration. Es entsteht Aufwand, den man eigentlich erst investieren möchte, nachdem man sich endgültig für eine Migration entschieden hat.

Als Alternative zu diesem Vorgehen wünscht man sich ein System, das die Frage nach der Kompatibilität selbstständig entscheidet, ohne dass hierzu ein Entwickler benötigt wird. Da das selbstständige Entscheiden in der Praxis wohl kaum zu lösen sein wird, fokussieren wir uns in dieser Arbeit darauf, durch automatische Analyseverfahren mögliche Einschränkungen der Kompatibilität zu bestimmen, die einem Projektleiter als belastbare Grundlage dienen, die anstehende Migration zu planen und im Sinne einer Aufwandschätzung zu bewerten.

Im folgenden Abschnitt 2 werden wir unseren Ansatz[1], den wir im Rahmen der Entwicklung eines Software-Stacks für semantische Content-Management-Systeme entworfen haben, detailliert erläutern und verwandte Arbeiten betrachten. Unser Ansatz verwendet eine Kombination aus statischer Codeanalyse in Zusammenspiel mit der Auswertung von Frameworkdokumentationen basierend auf dem “Framework Description Meta-Model” (FDMM) [CBE+10]. Im Abschnitt 3 werden wir das FDMM kurz vorstellen, um dann in Abschnitt 4 auf die Kompatibilitätsanalyse einzugehen. Abschnitt 5 ist der Beschreibung unserer Implementierung vorbehalten. Wir schließen diese Arbeit mit einer kurzen Diskussion und einem Ausblick in Abschnitt 6.

[1]This work has been part-funded by the European Commission under grant agreement FP7-ICT-2007-3/ No. 231527 (IKS - Interactive Knowledge Stack).

2 Ansatz

Ein Hauptgrund für die gängige Praxis einer “Migration auf Probe” bei Frameworkaktualisierungen, ist die meist mangelhafte Dokumentation von Frameworks bzw., falls eine ausreichende Dokumentation vorliegt, das Fehlen einer Möglichkeit diese Informationen automatisch auswerten zu können. Aus diesem Grund haben wir in einem ersten Schritt ein Meta-Modell für Frameworkdokumentationen (FDMM) entwickelt, das in Abschnitt 3 genauer betrachtet wird. Die Motivation zur Erstellung eines neuen Meta-Modells war, dass die existierenden Ansätze in diesem Bereich zum einen nicht alle aus unserer Sicht erforderlichen Aspekte einer Frameworkdokumentation abdecken konnten und zum anderen, dass es keinen Ansatz gab, der Informationen so erfassen konnte, dass diese für Analyseprozesse automatisch verarbeitbar gewesen wären. Durch unsere Erfahrungen in der Meta-Modellierung entschlossen wir uns, die etablierten Ideen aus bestehenden Ansätzen mit erweiterten Möglichkeiten in ein vereinheitlichendes Modell zu überführen. Das FDMM definiert eine Sprache für Frameworkdokumentationen und macht Vorgaben an Struktur und Inhalt. Das Meta-Modell als formale Grundlage dieser Sprachdefinition gestattet es uns, eine FDMM-basierte Frameworkdokumentation automatisch verarbeiten zu können.

Da die Erstellung einer entsprechenden Dokumentation, obwohl in Forschung und Praxis als sehr wichtig angesehen [Par11], eine zu häufig unbeachtete Aufgabe ist, kann man hinterfragen, ob in der Praxis eine FDMM-basierte Dokumentation erstellt werden wird. Wir glauben jedoch, dass erst die Auswertbarkeit der Informationen für einen Anwendungszweck, wie den hier beschriebenen, den Wille zur Dokumentation steigern wird. Mit dem FDMM können wir zeigen wie Meta-Modell basierte Sprachen helfen, Informationen weiterverarbeiten zu können und so Mehrwerte zu erzeugen.

Unser Lösungsansatz in Abbildung 3 basiert darauf, dass beide Frameworkversionen mit Hilfe des FDMM dokumentiert wurden. Die Dokumentation wird verwendet, um durch Vergleich die Unterschiede zwischen zwei Frameworkversionen zu extrahieren. Die Differenz zwischen den Dokumentationen, gemeinsam mit Informationen über die Benutzung, ermöglicht uns eine neue Form der Bewertung der Kompatibilitätsfrage.

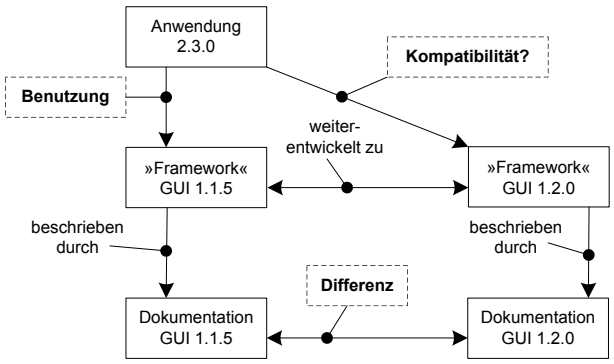


Abbildung 3: Lösungsansatz

Neben einer FDMM-basierten Dokumentation des Frameworks benötigen wir Informationen darüber, welche Teile des Frameworks von der Anwendung benutzt werden. Diese Benutzung könnte ebenfalls in auswertbarer Form dokumentiert worden sein. Wir gehen jedoch davon aus, dass eine solche Information nicht immer vorliegen wird, so dass wir statische Codeanalysen zur Analys der Bezüge zwischen Anwendung und Framework einsetzen. Diese Bezüge entstehen, weil z.B. eine Erweiterung in der Anwendung eine Schnittstelle implementiert, die von einem Framework definiert wird. Dies ist ein Hinweis darauf, dass die Anwendung an dieser Stelle einen Erweiterungspunkt des Frameworks benutzt. Neben der Analyse des Quellcodes können zusätzlich noch Ressourcen wie Konfigurationsdateien analysiert werden. Ziel dieser *Benutzungsanalyse* ist herauszufinden, welche Erweiterungspunkte des Frameworks von der Anwendung benutzt werden.

Parallel zur Benutzungsanalyse können wir die FDMM-basierte Dokumentationen beider Versionen in einer *Differenzanalyse* miteinander vergleichen und Unterschiede ermitteln. Anschließend können wir durch Kombination der Ergebnisse Rückschlüsse auf die Kompatibilität ziehen. Hierzu führen wir eine *Kompatibilitätsanalyse* in Bezug auf die verwendeten Erweiterungspunkte durch und bewerten gefundene Unterschiede im Hinblick auf zu erwartende Kompatibilitätsprobleme. Die Bewertung während der Kompatibilitätsanalyse liefert als Ergebnis eine Liste von Einschränkungen, die die Kompatibilität entweder potentiell oder in jedem Fall brechen. Potentiell brechende Einschränkungen sind jene, bei denen ein Bruch der Kompatibilität vorliegen könnte, wenn zum Beispiel das Framework durch neue Funktionen erweitert wird. Ein Beispiel für eine brechende Einschränkungen ist das Löschen eines bisher verwendeten Erweiterungspunktes.

Im Ergebnis dieser Analyseschritte, die in Abbildung 4 noch einmal dargestellt sind, liefert unser Ansatz eine Liste von Einschränkungen, die zum Beispiel von einem Projektleiter dazu verwendet werden kann, um die Auswirkungen einer Frameworkaktualisierung beurteilen zu können. Diese Auswirkungen können wiederum im Sinne von Aufwand bewertet werden, so dass man ohne Ausprobieren, eine Entscheidungsgrundlage hat, um für oder gegen eine Aktualisierung zu sein.

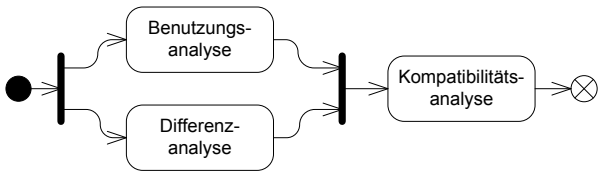


Abbildung 4: Analyseprozess

Das Thema Kompatibilität geht bereits auf Arbeiten aus dem Jahr 1968 [Gos68] zurück und hat viele Facetten. Verwandte Arbeiten für den Bereich der Kompatibilitätsanalyse beziehen sich meist auf das Problem der Kompatibilität von Komponenten und Services. Hier werden bereitgestellte und benötigte Schnittstellen betrachtet, um Aussagen über die Kompatibilität zu treffen, z.B. in [CZ06]. Eine Übersicht existierender Ansätze bietet [Cla06], wobei insbesondere die Arbeit von McCamant und Ernst [ME04] zu benennen ist, die Aussagen zur Kompatibilität bei Aktualisierung einer Komponente darüber ableiten, dass

sie das Verhalten beider Komponenten auf Basis von Tests protokollieren. Unser Ansatz geht davon aus, dass eine valide Dokumentation eine bessere Quelle für auswertbare Informationen darstellt, da in einer Dokumentation leichter alle Veränderungen erfasst werden können, als dass diese durch Tests und Beobachtung aufgedeckt werden könnten.

3 Meta-Modell zur Dokumentation von Frameworks

Für die Modellierung des FDMM zur Dokumentation von Frameworks haben wir nach eingehender Analyse existierender Frameworks und bestehender Ansätze zur Frameworkdokumentation eine Reihe von Anforderungen an Frameworkdokumentationssprachen ermittelt, die mit dem FDMM umgesetzt wurden. Ein Auszug der in [CBE+10] gestellten Anforderungen, die in Form von Merkmalen bezüglich einer relevanten Fragestellung aufgenommen wurden, zeigt Tabelle 1. Wir beschränken uns hier auf die Merkmale, die für eine Differenzanalyse im Hinblick auf die Kompatibilität von Interesse sind.

ID	Merkmal	Fragestellung
M1	Statische und dynamische Architekturübersicht	Wie ist die Gesamtarchitektur des Frameworks und an welchen Stellen befinden sich Erweiterungspunkte? Wie ist der Prozessfluss zwischen Framework und Erweiterungen?
M2	Erweiterungspunkt	Welche Erweiterungspunkte bietet das Framework an welchen Stellen in der Gesamtarchitektur und wie können diese genutzt werden?
M2.1	Bindung	Wie entsteht die Bindung zwischen einer Erweiterung und dem Framework, so dass die Erweiterung entsprechend eingebunden und verwendet wird?
M2.2	Installation	Wie und in welchem Format muss ein Erweiterung im Framework installiert (engl. deployed) werden?
M2.3	Protokoll	Welches Schnittstellenprotokoll muss eingehalten und bei der eigenen Implementierung einer Erweiterung berücksichtigt werden?

Tabelle 1: Dokumentationsmerkmal

Verwandte Ansätze im Bereich der Frameworkdokumentation arbeiten problemorientiert nach dem Kochbuchprinzip [SSP95, MCK97, OCM99], wobei ein häufig auftretendes Problem beschrieben und dann ein Rezept für dessen Lösung mit Hilfe des Frameworks vorgeschlagen wird. Ein Problem bei diesem Vorgehen ist, dass nicht jedes Problem in einer solchen Sammlung verfügbar ist und man bei Abweichungen vom Problem eine andere Form von Referenz vermisst. Diese Form von Unvollständigkeit zusammen mit dem Problem, dass es keine automatisch auswertbare Struktur in diesen Rezepten gibt, machen diese Ansätze unbrauchbar für eine Differenzanalyse, wie sie hier angestrebt wird.

Andere Ansätze entwickelten eigene Sprachen für die Dokumentation. Beispiele sind UML-basierte [OMG10] Sprachen wie die “UML-F” [FPR00, FPR02] – ein UML-Profil zur Dokumentation von Frameworkarchitekturen, die “UML-FD” [LSTM06] oder die “F-UML”

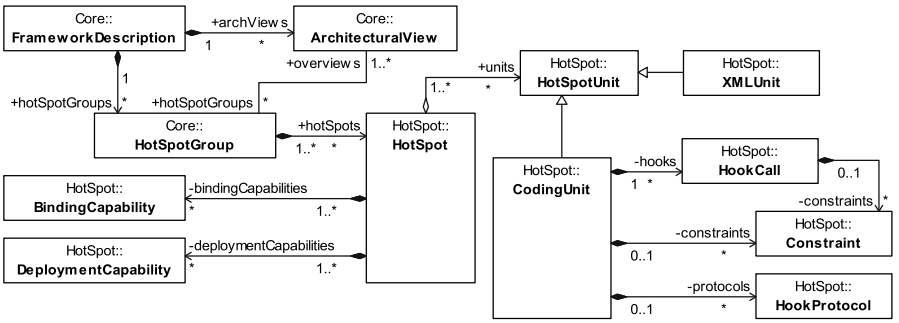


Abbildung 5: Auszug aus dem FDMM

[BBAGH03]. All diese Sprachen haben den Nachteil, dass sie nach den in Tabelle 1 aufgestellten Anforderungen unvollständig sind (siehe [CBE+10] für eine vergleichende Darstellung). Aus diesen Gründen haben wir das nachfolgend beschriebene FDMM entwickelt, das viele Ideen bestehender Ansätze in einem neuen Entwurf vereint und außerdem alle relevanten Merkmale einer Frameworkdokumentation abdeckt.

Das FDMM ist ein modular aufgebautes MOF-Modell [OMG06] dessen Kern aus drei Paketen besteht. Wir wollen hier aus Platzgründen nur die elementaren Teile des FDMM, dargestellt in Abbildung 5, kurz erläutern und den Bezug zu den Anforderungen aus Tabelle 1 herstellen. Im Paket `Basis` werden einige grundlegende Elemente definiert, wie eine Basisklasse für alle benannten Elemente. Das `Core` Paket enthält die Kernelemente von Frameworkdokumentationssprachen. Neben dem Wurzelelement `FrameworkDescription` gehören hierzu Sichten auf die Frameworkarchitektur, die über das Element `ArchitecturalView` (M1) repräsentiert werden. Erweiterungspunkte (`HotSpot`) werden gruppiert (`HotSpotGroup`) und mit der Architektur verlinkt.

Die Details eines `HotSpots` (M2) sind im gleichnamigen dritten Paket spezifiziert. Zu jedem `HotSpot` kann die Umsetzung des Bindens (`BindingCapability`) (M2.1) an das Framework und des Installierens (`DeploymentCapability`) (M2.2) im Framework dokumentiert werden. Ein `HotSpot` besteht aus einer Reihe von Untereinheiten (`HotSpotUnit`), die weitere Details zur Benutzung dokumentieren. Eine spezielle Form sind die `CodingUnits`, die zu implementierende Anteile beschreiben. Neben `CodingUnits` sind auch andere Formen denkbar, bei denen man zum Beispiel eine Erweiterung erstellt, indem man nicht im eigentlichen Sinne programmiert, sondern etwa eine XML-Datei mit entsprechendem Inhalt anlegt (`XMLUnit`). Eine `CodingUnit` beschreibt eine Reihe von zu implementierenden Methoden, genannt `HookCalls`, die vom Framework aufgerufen werden, um die Erweiterung zu bedienen. Solche `HookCalls` und die `CodingUnit` selbst können einer Reihe von Einschränkungen (`Constraints`) unterliegen, die zum Beispiel den Wertebereich der Parameter eines `HookCalls` einschränken. Eine weitere wichtige Eigenschaft ist das Kommunikationsprotokoll (`HookProtocol`) (M2.3) zwischen einem Erweiterungspunkt und einer Erweiterung beschreiben zu können. Es gilt zu beachten, dass hinter den hier gezeigten Elementen, wie einem `HookCall`

oder einem `HookProtocol`, sich weitere Elemente befinden. Um etwa die Details eines `HookProtocols` beschreiben zu können, benötigt man eine eigene Sprache wie UML-Protokollzustandsautomaten, die in das FDMM integriert werden kann. Für weitere Details verweisen wir auf [CBE+10]. Ein Dokumentationsbeispiel werden wir im nachfolgenden Abschnitt 4 geben.

4 Kompatibilitätsanalyse

Die Kompatibilitätsanalyse verwendet die Informationen aus Benutzungs- und Differenzanalyse. Die Benutzungsanalyse liefert Bezüge zwischen Anwendung und Framework, so dass die verwendeten Erweiterungspunkte identifiziert werden können. Die Differenzanalyse ermittelt alle Unterschiede zwischen zwei Versionen der Frameworkdokumentation. In der Kompatibilitätsanalyse wird geprüft, ob sich Änderungen bei verwendeten Erweiterungspunkten ergeben haben. Diese Änderungen werden im Hinblick auf die Kompatibilität bewertet und eine Liste von Einschränkungen der Kompatibilität generiert.

Wir werden dieses Vorgehen an einem vereinfachten Praxisbeispiel aus dem Bereich Web-Anwendungen verdeutlichen. Das Open-Source Framework "Apache MyFaces"[2] ist eine Implementierung der "Java Server Faces" (JSF) Spezifikation[3]. Mit diesem Framework lassen sich zum Beispiel GUI-Elemente wie Listen, Tabellen und Formulare leicht auf einer Web-Seite darstellen. Um eigene GUI-Elemente zu entwerfen bietet das Framework den Erweiterungspunkt `HS-UIComponent` an. Die Dokumentation in Abbildung 6 für MyFaces Version 1.1.5 besagt, dass man für diesen Erweiterungspunkt von der Klasse `UIComponent` erben. Mit der Vererbung muss man zusätzlich die Hook-Methoden `getValueBinding()` und `getAttributes()` für das eigene GUI-Element überschreiben, um das gewünschte Verhalten zu implementieren.

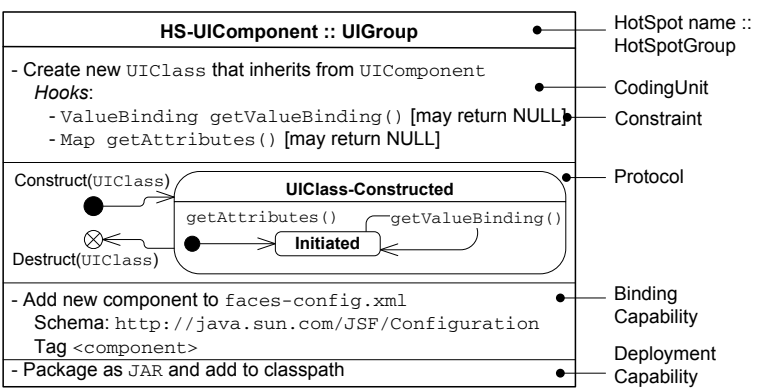


Abbildung 6: HotSpot HS-UIComponent in MyFaces 1.1.5

[2] http://myfaces.apache.org
[3] http://javaserverfaces.java.net/

Constraints der Hook-Methode `getAttributes()` besagen zusätzlich, dass diese Methode eine `Map` oder `NULL` zurückgeben muss.

Nehmen wir an, unsere fiktive Anwendung implementiert eigene GUI-Komponenten auf diese Weise. Die Benutzungsanalyse erkennt, dass es in der Anwendung Klassen gibt, die von `UIComponent` erben. Es wird erkannt, dass `UIComponent` zum Erweiterungspunkt `HSUIComponent` gehört. Nun verändern wir die Version von MyFaces 1.1.5 auf 1.2.0. In der neueren Version haben sich einige Dinge verändert, wie man dem Auszug aus der Dokumentation in Abbildung 7 entnehmen kann.

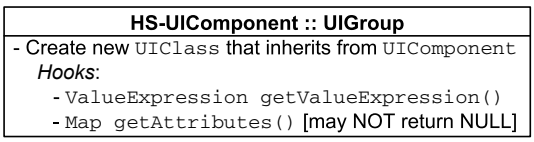


Abbildung 7: HotSpot HS-UIComponent in MyFaces 1.2.0

Wir führen eine Differenzanalyse durch und erhalten das Analyseergebnis in Tabelle 2. Es zeigt Ereignisse wie neue, gelöschte oder veränderte Elemente in der Dokumentation und einen Hinweistext zur späteren Auswertung. Wir sehen, dass wir in unseren GUI-Komponenten die Methode `getValueBinding()` nicht mehr verwenden dürfen und die neue Hook-Methode `getValueExpression()` zu berücksichtigen ist. Darüber hinaus können z.B. auch Änderungen der Constraints erkannt werden, so dass man nun weiß, dass die Hook-Methode `getAttributes()` in Version 1.2.0 nicht mehr den Wert `NULL` zurückgeben darf. Auch hier nicht gezeigte Veränderungen am Kommunikationsprotokoll zwischen Framework und Erweiterung können so identifiziert werden.

Ereignis	Hinweis
Neu	Der Erweiterungspunkt 'HS-UIComponent' verwendet nun die neue Hook-Methode 'getValueExpression()'.
Löschung	Der Erweiterungspunkt 'HS-UIComponent' verwendet die Hook-Methode 'getValueBinding()' nicht mehr'.
Änderung	Der Constraint der Hook-Methode 'getAttributes()' wurde geändert. *1.1.5*: 'Die Methode muss eine Map oder NULL zurückgeben.' *1.2.0*: 'Die Methode muss eine Map zurückgeben – nicht NULL.'

Tabelle 2: Ergebnis einer Differenzanalyse

Die Differenzanalyse ermittelt sämtliche Veränderungen an der Frameworkdokumentation. In einer abschließenden Kompatibilitätsanalyse würde man nun das Ergebnis der Differenzanalyse bezüglich verwendeter Erweiterungspunkte betrachten. Während dieser Analyse können die Aussagen hinsichtlich potentiell brechender oder brechender Einschränkungen bewertet werden. Die Löschung der Hook-Methode `getValueBinding()` ist zum Beispiel eine brechende Einschränkung. Eine Veränderung des Constraints ist potentiell brechend. Das Ergebnis der Kompatibilitätsanalyse ist eine bewertete Liste aller relevanten Veränderungen.

5 Implementierung

Wir setzen unseren Ansatz für Java-basierte Frameworks um, da Java eine häufig eingesetzte Technologie bei der Entwicklung betrieblicher Informationssysteme ist. Zur Erstellung einer FDMM-basierten Frameworkdokumentation wird das FDMM mit Hilfe des "Eclipse Modeling Frameworks" (EMF) [SBPM09] umgesetzt. Diese EMF-Umsetzung des FDMM ist die Grundlage für die Implementierung eines Editors, der sich in die Entwicklungsumgebung Eclipse integriert und so die parallele Dokumentation eines Frameworks während seiner Entstehung ermöglicht.

Zur Differenzanalyse planen wir das Werkzeug EMF-Compare[4] einzusetzen, das es ermöglicht die Unterschiede zwischen zwei EMF-basierten Modellen, wie dem FDMM, zu bestimmen. Die Benutzungsanalyse basiert auf dem Open Source Werkzeuge "Dependency Finder"[5], das eine statische Codeanalyse auf Java-Bytecode-Ebene durchführt. Durch Erweiterungen dieses Werkzeuges können die Codebezüge einer Anwendung zu verwendeten Java-Archiven (JAR-Dateien) in einer Datenbank gespeichert werden, so dass diese einmal erhobenen Informationen für weitere Analysen zur Verfügung stehen. Java-Archive sind hierbei das technische Format, in dem Frameworks ausgeliefert werden, so dass wir mit dem Dependency Finder Informationen über die Benutzungsbeziehung zwischen Anwendung und Framework erhalten. Für die Zukunft sind Erweiterungen der Benutzungsanalyse geplant, die in der Lage sein werden zusätzlich Informationen aus Ressourcen wie Konfigurationsdateien auszuwerten.

Zur Umsetzung der Kompatibilitätsanalyse setzt ein eigenes Werkzeug die gewonnenen Informationen aus der Benutzungsanalyse mit den Informationen der Differenzanalyse in Beziehung. Für jede Benutzung eines Erweiterungspunktes des Frameworks prüfen wir, ob sich laut Frameworkdokumentation etwas an der Umsetzung dieses Erweiterungspunktes verändert hat. Ist dies der Fall, wird die Veränderung entsprechend bewertet und eine Aussage zur Kompatibilitätseinschränkung generiert. Die Liste der ermittelten Einschränkungen in Form einer Textdatei bildet die Ausgabe der Kompatibilitätsanalyse.

6 Diskussion und Ausblick

Für die praktische Anwendbarkeit unseres Ansatzes arbeiten wir zunächst daran die beschriebene Werkzeugkette stabil zu implementieren und Erfahrungen durch praktische Evaluationen zu sammeln. Auf Forschungsseite wollen wir das hier beschriebene Problem zwischen einer Anwendung und einem Framework auf die Beziehung zwischen Anwendung und beliebig vielen Frameworks ausweiten. Eine komplexere Anwendung, skizziert in Abbildung 8, verwendet z.B. in jeder Anwendungsschicht spezialisierte Frameworks, die an ihren Erweiterungspunkten miteinander interagieren. Darüber hinaus verwendet jedes Framework eine Reihe von Bibliotheken, um z.B. XML-Dokumente verarbeiten zu können oder Log-Meldungen zu generieren. Hinzu kommen die Anteile eigener Erwei-

[4] http://wiki.eclipse.org/EMF_Compare
[5] http://depfind.sourceforge.net

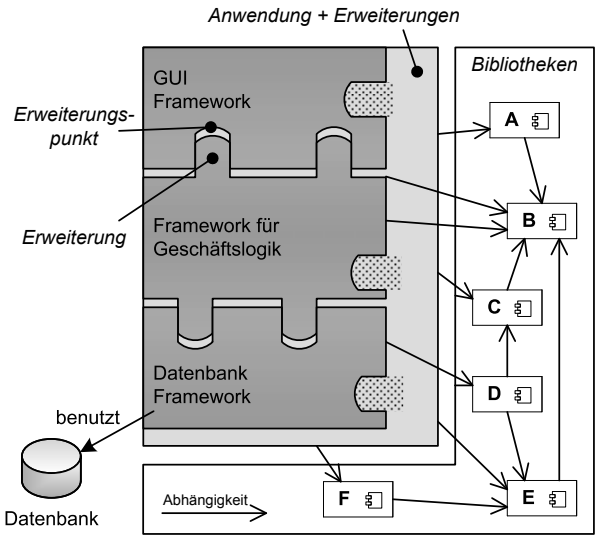


Abbildung 8: Skizze einer komplexen Anwendung

terungen der Frameworks plus benötigter eigener Bibliotheken, wobei auch Bibliotheken untereinander Abhängigkeiten haben können. Es entsteht ein komplexes Abhängigkeitsgeflecht, dass es bei der Aktualisierung eines Frameworks zu beachten gilt. Es gilt also die Kompatibilitätsanalyse auf mehrere Frameworks plus abhängiger Bibliotheken ausdehnen zu können. In diesem Zusammenhang planen wir Verbesserungen für die Integration von Benutzungs-, Differenz- und Kompatibilitätsanalyse in eine produktreifere Werkzeugkette, die als Open-Source-Software in Zukunft auch einmal in der industriellen Praxis Verwendung finden kann.

Literatur

[BBAGH03] N. Bouassida, H. Ben-Abdallah, F. Gargouri und A. B. Hamadou. Formalizing the framework design language F-UML. In *Proceedings of SEFM'03*, Seiten 164–172. IEEE, September 2003.

[CBE+10] Fabian Christ, Jan-Christopher Bals, Gregor Engels, Christian Gerth und Markus Luckey. A Generic Meta-Model-based Approach for Specifying Framework Functionality and Usage. In *Proceedings of TOOLS Europe'10*, Jgg. 6141 of *LNCS*, Seiten 21–40. Springer, 2010.

[Cla06] Program Compatibility Approaches. In Frank de Boer, Marcello Bonsangue, Susanne Graf und Willem-Paul de Roever, Hrsg., *Formal Methods for Components and Objects*, Jgg. 4111 of *LNCS*, Seiten 243–258. Springer, 2006.

[CZ06] D.C. Craig und W.M. Zuberek. Compatibility of Software Components - Modeling and Verification. In *Proceedings of the International Conference on Dependability of Computer Systems (DepCos-RELCOMEX'06)*, Seiten 11–18, May 2006.

[Deu89] L. P. Deutsch. Design reuse and frameworks in the Smalltalk-80 system. *Software reusability*, 2:57–71, 1989.

[FPR00] Marcus Fontoura, Wolfgang Pree und Bernhard Rumpe. UML-F: A Modeling Language for Object-Oriented Frameworks. In *Proceedings of the European Conference on Object-Oriented Programing (ECOOP2000), Sophia Antipolis and Cannes, France*, June 2000.

[FPR02] Marcus Fontoura, Wolfgang Pree und Bernhard Rumpe. *The UML Profile for Framework Architectures*. Addison Wesley, 2002.

[Gos68] John A. Gosden. Software compatibility: what was promised, what we have, what we need. In *Proceedings of the fall joint computer conference, part I (AFIPS '68)*, Seiten 81–87, New York, NY, USA, 1968. ACM.

[JF88] Ralph E. Johnson und Brian Foote. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2):22–35, June/July 1988.

[Joh97] Ralph E. Johnson. Frameworks = (Components + Patterns). *Commun. ACM*, 40(10):39–42, 1997.

[LSTM06] Srgio Lopes, Carlos Silva, Adriano Tavares und Joo Monteiro. Describing Framework Static Structure: promoting interfaces with UML annotations. In *Proceedings of the WCOP'06*, Seiten 54–61. Universität Karlsruhe - Fakultät für Informatik, July 2006. ISSN 1432 - 7864.

[MCK97] Matthias Meusel, Krzysztof Czarnecki und Wolfgang Köpf. A model for structuring user documentation of object-oriented frameworks using patterns and hypertext. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97)*, Jgg. 1241 of *LNCS*, Seiten 496–510. Springer, 1997.

[ME04] Stephan McCamant und Michael D. Ernst. Early Identification of Incompatibilities in Multi-component Upgrades. In *Proceedings of the ECOOP'04, Oslo (Norway)*, Jgg. 3086 of *LNCS*, Seiten 117–140. Springer, 2004.

[OCM99] A. Ortigosa, M. Campo und R. Moriyn. Enhancing framework usability through smart documentation. In *Proceedings of the Argentinian Symposium on Object Orientation*, Seiten 103–117, Buenos Aires, Argentina, 1999.

[OMG06] OMG. Meta Object Facility (MOF) Core Specification, V2.0. Bericht, OMG, 2006.

[OMG10] OMG. Unified Modeling Language (OMG UML), Superstructure, V2.3, May 2010.

[Par11] David Lorge Parnas. Precise Documentation: The Key to Better Software. In Sebastian Nanz, Hrsg., *The Future of Software Engineering*, Seiten 125–148. Springer, 2011.

[SBPM09] Dave Steinberg, Frank Budinsky, Marcelo Paternostro und Ed Merks. *EMF - Eclipse Modeling Framework*. Addison Wesley, second edition. Auflage, 2009.

[Sch97] Han Albrecht Schmid. Systematic framework design by generalization. *Commun. ACM*, 40(10):48–51, 1997.

[SSP95] Albert Schappert, Peter Sommerlad und Wolfgang Pree. Automated support for software development with frameworks. In *Proceedings of the Symposium on Software reusability (SSR '95)*, Seiten 123–127, New York, NY, USA, 1995. ACM Press.

Enriching OSGi Service Interfaces with Formal Sequential Contracts*

Marco Müller, Moritz Balz, Michael Goedicke
paluno - The Ruhr Institute for Software Technology
University of Duisburg-Essen, Campus Essen
Essen, Germany
{marco.mueller, moritz.balz, michael.goedicke}@paluno.uni-due.de

Abstract: Architecture description languages define component interfaces with sequential contracts, which allow for static analysis of method call sequences in component or service interactions. However, component and service platforms like OSGi for Java do not provide mechanisms for the specification or enforcement of such sequential contracts. Thus the contracts are only defined in the documentation which might be outdated when long-living systems evolve at the implementation level. This vision paper proposes to attach formal sequential models, in our case *interface automata*, to the interface definition of OSGi services, so that the modeling information is permanently and tightly coupled to the implementation. This enables consistent documentation, static analysis of component interactions at design time, and real-time enforcement of behavioural contracts at run time. By this means, component interactions can be seamlessly verified in long-living systems when components and their connections are added or changed over time.

1 Motivation

In component-based and service-oriented software architectures, systems consist of components which are represented to their context by interfaces that describe their functionality. The implementation of this functionality is hidden to the environment, and accessed through the published interfaces. Beugnard et al. [BJPW99] define four levels of interface descriptions: (1) *syntactic* level, i.e. method signatures; (2) *behavioural* level, i.e. pre- and postconditions for method invocations; (3) *synchronization* level, which includes call synchronization and sequences; and (4) *quality of service* level, including performance and security information. Interface definition languages of current programming languages usually allow for defining syntactic level interfaces, i.e. the methods available for invocation. For defining behaviour level interfaces, programming language extensions like the Java Modeling Language (JML) [LC05] can be used.

Components often assume that their methods are called in certain orders, e.g. when a component requires authentication before permitting the invocation of further methods.

*Part of the work reported herein was funded by the German Federal Ministry of Education and Research under the grant number 03FPB00320.

While these synchronization level interface descriptions are important for the interface's users, they are usually not explicitly and formally stated in current interface descriptions of programming languages like Java and their component and service frameworks, like OSGi [OSG05]. Hence, instead of focusing on the business aspects only, methods must validate the current state to prevent errors and possibly inconsistent data. This is usually performed at the beginning of the code in the method body, in which object attributes are inspected for reasonable values, and exceptions are thrown when an invalid state was identified.

Several formal techniques exist for describing sequential contracts of interfaces. As an instance of them, *interface automata* [dAH01] define permitted call sequences based on state machines. These models and related tools can be used for verifying component or service interaction with respect to the sequential description at design time. In this case, the models reside in the software documentation. However, when designing architectures for long-living software, it must be considered that the documentation might be unavailable or become inconsistent when the software evolves. In this contribution we propose to attach interface automata directly to OSGi service interface descriptions, thus enriching the component interfaces with *synchronization level* information. At design time, the models can be used for static analysis regarding component interactions. At run time, the model information attached to the interfaces can be used to ensure that only valid sequences of method calls are performed. This is of interest especially for system evolution, when components are changed or added that are not known beforehand, since valid interaction sequences can be ensured permanently once they are defined in the program code.

This paper is organized as follows: In section 2 we show how to enrich OSGi service interfaces with interface automata definitions. The use of these models at design time and run time is presented in section 3. A short overview of related work is given in section 4 before the paper is concluded in section 5.

2 Approach

Interface automata [dAH01] are essentially finite state machines with input, output, and internal actions, where each input defines a received method call and each output defines an outgoing method call. Systems of interface automata communicate via input and output actions, so that service and component interactions can be described with sequential contracts. Methods of interfaces are input actions that are executed when the method is called. These actions trigger transitions (called *steps*) in the automaton. If the interface automaton has a step with the called method in the current state, the call is permitted.

OSGi services are components that can be described with interface automata, since they publish functionality with a Java interface and thus a set of methods. To reach our objective of coupling sequential modeling information to these interfaces, we use meta data annotations in Java source code [Sun04]: The annotation `@InterfaceAutomaton` can be attached to OSGi service interfaces and contains information about the sets of states and steps. Each state is represented by a nested annotation `@State` denoting the state's name. One of the states is marked as initial, representing the entry point of the sequential

```
@InterfaceAutomaton(
  states={ @State(value="loggedOut",initial=true),
           @State("loggedIn") },
  steps={
    @Step(from="loggedOut", action="login(java.lang.String,java.lang.String)",
        to="loggedIn"),
    @Step(from="loggedIn", action="execute(java.lang.String)", to="loggedIn"),
    @Step(from="loggedIn", action="logout()", to="loggedOut")} )
public interface IShell {
  public void login(String user, String password);
  public void logout();
  public String execute(String command);
}
```

Listing 1: An interface automaton definition for a Java interface

contract. Each step is represented by a nested annotation `@Step` referencing an originating state, a Java method signature as input action, and a target state. Since we consider provided interfaces only, each action is an input action in this context.

Listing 1 shows a Java interface with an interface automaton annotation. This automaton is defined by 2 states `loggedOut` and `loggedIn`, 3 input actions (the interface method definitions), and 3 corresponding steps. The contract defines that the first method to be called is `login`, before the methods `execute` and finally `logout` may be called.

3 Usage of the Models

When interface automata are attached to OSGi services as described above, component interaction can be verified with appropriate tools, thus ensuring valid invocation sequences even if the participating components change. The usage of annotations for embedding the interface automaton into the program code has several advantages: Annotations are typed meta data that can be accessed programmatically at design time and run time of the software, as the automaton definition is directly embedded into the source code. Thus the modeling information can be used for different kinds of verification.

3.1 Design Time

In OSGi, some Java interfaces are service descriptions used by the framework during execution. Since the models are not only filed in the documentation, but coupled to these service descriptions, they are always available with the source code when long-living systems evolve, even when the documentation is outdated. The static nature of annotations allows for graphical editing for an easier understanding of more complex automata because they can be read and written by an according editor as any other notation.

Since the Java methods define the actions in the source code unambiguously, it is possible to perform static code analysis for component interactions. This requires to extract models

that describe how services are used by other components. Approaches for this already exist, e.g. in the JADET tool [WZL07]; for our purpose, they must be adapted to extract calls to OSGi service interfaces from the control flow to create an interface automaton with output actions, which can then be matched with the automaton describing the service.

3.2 Run Time

Since the annotations are accessible at run time, they can be used in the OSGi Service Platform for monitoring and enforcing the compliance of calls to service interfaces with their behavioural contract. To accomplish this, a technique must exist that can observe and prevent method calls. Examples for such techniques are aspect-oriented programming (AOP) [KLM+97] and dynamic proxies [Sun10].

As a proof-of-concept[1] we implemented the integration for the OSGi implementation Eclipse Equinox [Ecl11a] in combination with Equinox Aspects [Ecl11b], which enables AOP in an OSGi environment. This proof-of-concept includes business components as services which are described by interfaces with interface automata definitions, and one *observer* component which observes and possibly rejects interface calls using AOP. The observer reads interface automata from the annotations first when service components with interface automata descriptions are deployed. Subsequently, all method calls to these interfaces are verified with respect to the model. When the sequential contract is broken, the observer will reject calls.

Figure 1 visualizes this concept. The left side shows the interaction of two business components without the explicitly modeled sequential contract. As explained in the motivation, the business components must themselves check the permission of the call. The right side shows the same scenario with the proposed concept in use: The observer is notified about method calls, verifies them against the interface automaton, and rejects the second `login` method, since it is not valid in the example interface automaton shown in section 2.

4 Related Work

The extension of interfaces with further information is subject to many publications. However, most of them focus on implementing behavioural contracts, i.e. pre- and postconditions for method calls. It is possible to emulate call sequence specifications using behavioural descriptions with dedicated internal state fields that are checked in preconditions and set in postconditions of method invocations. This approach is argued to be error-prone and hard to understand since the sequential contract is modeled indirectly [CP07].

Heinlein [Hei09] describes concurrency and sequence constraints for Java classes using interaction expressions. The constraints are checked at run time, postponing prohibited method calls. However, the Java language is extended in this approach to integrate the

[1] The prototype can be downloaded at http://www.s3.uni-duisburg-essen.de/index.html?op=view&id=342

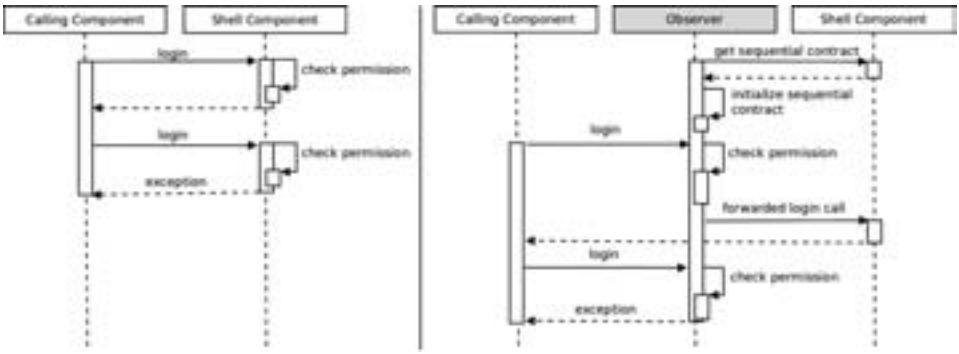


Figure 1: On the left side, the callee needs to check whether the calls respect the sequential contract. On the right side, these checks are processed by the contract enforcing component. The callee can focus on the business issues.

checks for the constraints. Our approach does not require changes in the language, and is thus easier to adopt. Pavel et al. [PNPR05] introduce a framework which allows to check and reject method calls to Enterprise JavaBeans [Sun08] components based on Symbolic Transition Systems. In their approach, the state checks have to be implemented manually or with a precompiler directly into the business methods. In contrast to our approach, the validation code is integrated in the business methods, which makes them harder to read and understand, especially when maintaining or evolving a legacy software.

5 Conclusion and Future Work

In current programming languages and frameworks, component or service interfaces are usually described structurally. This only includes the method signatures. However, components often assume certain orders of method calls, e.g. an authentication before further methods may be called. While several formal techniques exist that allow for describing call sequences, these concepts are not reflected in current programming languages and frameworks. In this contribution we presented a concept for representing sequential contracts in Java for checking the compliance of a callers behaviour at run time, and for rejecting calls that break the sequential contract. We also presented how to integrate this concept in existing frameworks, using the OSGi Service Platform for a proof-of-concept.

As a benefit of the implementation, the business methods do not need to validate the component's state, but the development can focus on the business requirements. This eases program comprehension in software maintenance and evolution especially in long-living systems, when documentation might be absent, because the sequential contracts are not defined in the documentation, but directly in the code.

As future work we plan to build tools for developing the automata graphically, synchronizing them with the code, and performing static analysis as introduced above. This allows for validation of the approach in large development projects. We will also improve the no-

tation for interface automata in code. In addition, we plan to integrate interface automata in further component frameworks and service platforms, e.g. Enterprise JavaBeans. As another step, we will statically verify the compliance of sequential contracts in complex architectures, using input and output interfaces for components.

References

[BJPW99] Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, and Damien Watkins. Making Components Contract Aware. *Computer*, 32(7):38–45, 1999.

[CP07] Yoonsik Cheon and Ashaveena Perumandla. Specifying and Checking Method Call Sequences of Java Programs. *Software Quality Journal*, 15(1):7–25, March 2007.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface Automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE), ACM*, pages 109–120, 2001.

[Ecl11a] Eclipse Foundation. Equinox, 2011. `http://www.eclipse.org/equinox/`.

[Ecl11b] Eclipse Foundation. Equinox Aspects, 2011. `http://eclipse.org/equinox/incubator/aspects/`.

[Hei09] Christian Heinlein. Advanced Thread Synchronization in Java Using Interaction Expressions. *Objects, Components, Architectures, Services, and Applications for a Networked World*, 2591/2009:345–365, 2009.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwing. Aspect-Oriented Programming. *ECOOP '97 - Object-Oriented Programming: 11th European Conference*, LNCS 1241:220–242, 1997.

[LC05] G.T. Leavens and Y. Cheon. Design by Contract with JML. *Draft, available from jmlspecs. org*, 1:4, 2005.

[OSG05] OSGi Alliance. *OSGi Service Platform, Core Specification, Release 4, Version 4.1*. IOS Press, Inc., 2005.

[PNPR05] Sebastian Pavel, Jacques Noye, Pascal Poizat, and Jean-Claude Royer. Java Implementation of a Component Model with Explicit Symbolic Protocols. In *In Proceedings of the 4th International Workshop on Software Composition (SC'05), volume 3628 of LNCS*, pages 115–124. Springer-Verlag, 2005.

[Sun04] Sun Microsystems, Inc. JSR 175: A Metadata Facility for the Java™Programming Language, 2004. `http://jcp.org/en/jsr/detail?id=175`.

[Sun08] Sun Microsystems, Inc. JSR 318: Enterprise JavaBeans™3.1 - Proposed Final Draft, March 2008. `http://jcp.org/en/jsr/detail?id=318`.

[Sun10] Sun Microsystems. Dynamic Proxy Classes, Java SE Documentation, January 2010. `http://java.sun.com/javase/6/docs/technotes/guides/reflection/proxy.html`.

[WZL07] Andrzej Wasylkowski, Andreas Zeller, and Christian Lindig. Detecting Object Usage Anomalies. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 35–44, New York, NY, USA, 2007. ACM.

Towards Software Sustainability Guidelines for Long-living Industrial Systems

Heiko Koziolek[1], Roland Weiss[1], Zoya Durdik[2], Johannes Stammel[2], Klaus Krogmann[2]

[1]Industrial Software Systems, ABB Corporate Research Ladenburg, Germany
[2]Forschungszentrum Informatik (FZI), Karlsruhe, Germany
heiko.koziolek@de.abb.com

Abstract: Long-living software systems are sustainable if they can be cost-effectively maintained and evolved over their complete life-cycle. Software-intensive systems in the industrial automation domain are typically long-living and cause high evolution costs, because of new customer requirements, technology changes, and failure reports. Many methods for sustainable software development have been proposed in the scientific literature, but most of them are not applied in industrial practice. We identified typical evolution scenarios in the industrial automation domain and conducted an extensive literature search to extract a number of guidelines for sustainable software development based on the methods found in literature. For validation purposes, we map one evolution scenario to these guidelines in this paper.

1 Introduction

Software systems in the industrial automation domain are typically *long-living* systems, i.e., some of them may operate for more than 20 years, because of the investment in the underlying machines and devices. Examples for such systems are distributed control systems to manage industrial processes, such as power generation, manufacturing, chemical production, or robotics systems to automate manual tasks, such as welding, pick&place, or sealing. Such systems are usually implemented with a variety of technologies and have very high requirements for safety, performance, and availability.

Because of their long life-cycles and complicated replacement procedures, industrial software systems are continuously maintained and evolved resulting in high costs, which amount for a large portion of the overall 10 billion dollar market. Thus, industrial automation companies, such as ABB, are interested in creating *sustainable* long-living software systems, i.e., systems that can be cost-effectively maintained over their complete life-cycle. This is a complex challenge, as industrial software systems are continuously subject to new requirements, new standards, failures, and technology changes during their operation time.

While a vast variety of methods for the cost-effective evolution of software systems have been proposed in the academic literature [CHK+01, GG08], it is difficult to select from these methods and to determine upfront whether they can address specific evolution prob-

lems. Ali Babar et al. [BG09] found in a survey that only 24% of the interrogated software architects were aware of scenario-based architecture analysis methods that can be used for the evaluation of evolution scenarios. A state-of-the-art overview and guidelines for software developers are missing, which could increase the willingness to apply the proposed, sophisticated methods in practice. For the industrial automation domain, a mapping from proven methods to typical evolution scenarios is required.

The contributions of this paper are (i) a list of typical evolution scenarios in the industrial automation domain, which can be used by third parties in evolution assessments and (ii) a sketch of initial sustainability guidelines based on an extensive literature survey. For the guidelines, we identified the most mature methods of sustainability and categorized them according to the development phases of a software system. In this paper, we provide a preliminary validation by mapping one sample evolution scenario to these guidelines to show that they are readily applicable in practice. A full validation of the guidelines is planned through several future case studies, which we briefly sketch in this paper.

The rest of this paper is structured as follows: Section 2 lists some re-occurring evolution scenarios in the industrial automation domain. Section 3 summarizes our preliminary guidelines for developing sustainable software systems structured against different development phases. Section 4 provides an initial validation of the guidelines by mapping them to one sample scenario and sketches planned validation activities. Finally, Section 5 discusses related approaches and surveys and Section 6 concludes the paper.

2 Evolution Scenarios in the Industrial Automation Domain

2.1 The Industrial Automation Domain

Industrial automation deals with the use of control systems and information technology to reduce manual work in the production of goods and services. While industrial automation systems originate from manufacturing, now there are many other application scenarios. Industrial control systems are for example used for power generation, traffic management, water management, pulp and paper handling, printing, metal handling, oil refinery, chemical processes, pharmaceutical manufacturing, or carrier ships.

Different types of software applications are used in industrial automation:

- Domain specific applications: e.g., control loops according to IEC 61131-3, typically below 50 KLOC
- Embedded software applications: e.g., running on controllers (e.g., ARM) interacting with custom hardware, programmed in C/C++, Assembler; OS: VxWorks, Linux, QNX, typically below 500 KLOC
- Large-scale software systems: e.g., distributed control systems (DCS), programmed in C++, C#, and Java; OS: Windows, Linux, millions lines of code

2.2 Evolution Scenarios

Industrial software systems have high requirements on reliability, safety, performance, security, and usability. Redundancy is often used on various levels to ensure reliability. Safety certifications are required in some domains for the controller software (e.g., according to IEC61508), which implies very high development standards. Performance issues are dangerous, because many systems are real-time critical, thus, missed deadlines imply a failure of the system and may harm human beings.

During their potentially more than 30 years life-cycles, these kinds of systems undergo various changes. Based on our experience with various software systems at ABB, we have identified a number of typical evolution scenarios. These scenarios highlight typical changes to the system and could be generalised to evolution scenario patterns. Future software systems could be evaluated with respect to these evolution scenarios. We have classified typical evolution scenarios for these systems according to ISO/IEC14765 into corrective, adaptive, perfective, and preventive scenarios. The following provides a selection of these scenarios.

Perfective scenarios modify a software product after delivery to improve for example the functionality, performance, or reliability. Some of the most frequent scenarios are listed below.

- new functionalities and services: customers continuously demand new features and functionality for industrial software (e.g., alarm management, history logging, optimizations)
- integration of third party components: special new functionality (e.g., certain data analysis functions) are sometimes already supported by third party vendors, whose components need to be integrated into an industrial system. Thus, writing adapters for these components is a common evolution scenario.
- integration of third party applications: industrial systems at customer sites often consist of software products from different vendors. Ensuring the interoperability between different applications is thus a continuous concern for industrial systems.
- integration of third party systems: industrial systems are often connected to manufacturing execution systems (MES) and/or enterprise resource planning systems (ERP). Their evolution often requires a corresponding evolution of the industrial software system
- safety certification: some parts of an industrial control system (e.g., controller devices) require an increasing safety certification (e.g., moving from IEC61508 SIL2 to SIL3) in certain application domains. For updated devices also the safety cases have to be updated or rebuilt.
- performance/scalability improvements: supporting more complex applications and processing more signals (e.g., measured in the maximum number of I/Os) from field devices is a latent requirement for industrial systems. For several software systems at ABB, a main feature of former revisions was the support for more devices and/or different devices.
- usability improvements: besides improving the layout of graphical user interfaces, also new functionality for improved usability is added to an industrial system, e.g., support

for multi-touch devices or multi-user support for engineering tools.
- security improvements: with the advent of service-oriented and web-based industrial control systems the amount of security issues has increased dramatically. Thus, there is a need to continuously supply security patches and enhancements to prevent attackers to exploit security weaknesses in the systems. The Stuxnet computer worm is a recent, highly publicized example, which required updated existing DCS systems.

Adaptive evolution scenarios refer to modification of a software product performed after delivery to keep a software product usable in a changed or changing environment. Examples of such scenarios are:

- new industry standards: industrial systems rely on multiple standards (e.g., OPC, IEC 61850), which are periodically revised or newly introduced. Compliance to these standards is thus a competitive factor and a re-occurring evolution scenario.
- migration to a new GUI framework: human machine interfaces are critical components in distributed control systems to allow operators efficiently running a plant. Replacing older frameworks (e.g., Visual Basic with WPF or Java AWT with SWT) is a re-occurring scenario and can have significant impacts on a system.
- migration to a new middleware: the communication between software components is often managed by a middleware, especially for distributed systems. New technologies or industry standards (e.g., OPC UA) require migration (e.g., from a COM-based communication to web services).
- support for new operating systems: operating systems typically have much shorter update cycles than industrial software systems. As industrial systems are often tightly integrated with operating systems, respective updates of the OS can require many modifications to the industrial software system.
- support for virtualization: for distributed control systems, new virtualization techniques allow saving server nodes and thus costs. Due to the real-time capabilities of the software special measures to support virtualization are required.
- support for multi-core processors: the general trend towards multi-threaded software is also present in the industrial domain. Increasing performance requirements due to more complex industrial processes lead to a desire to parallelize the software systems where possible to exploit multi-core CPUs.
- updated controller and field devices: many new devices are developed during the lifecycle of an industrial system. Their support is often critical to stay competitive.
- new network standards: while new network standards (e.g., Industrial Ethernet, WirelessHART) enable more efficient and fail-safe communication, they also often induce high migration efforts for existing systems

Corrective evolution scenarios refer to reactive modifications of a software product performed after delivery to correct discovered problems. In a former case study [KSB10], we analysed the bug tracking system of a large industrial control system and found that many bugs are still found by customers after release. The development of such a complex system cannot produce failure free software. While we could attribute failure reports to specific subsystems, we have not yet categorized the failure reports to derive bug-related

evolution scenarios, which remains future work..

Preventive evolution scenarios are modifications of a software product after delivery to detect and correct latent faults in the software product before they become effective faults. This may for example relate to refactoring or architecture improvements. From our current perspective in the industrial automation domain these activities similar to the activities in other domains.

3 Software Sustainability Guidelines

To tackle the evolution scenarios described in the former section and to ensure their cost-efficient implementation, we conducted a comprehensive literature survey [DKK+11] on analytical and corrective methods and tools for evolution issues. Our intent is to support developers in the industrial automation domain to be aware of state-of-the-art methods and their potential benefits and risks. This may allow a proactive approach towards sustainability and avoid firefighting evolution problems.

Our literature search targeted renowned journals and conference proceedings as well as books. We identified different kinds of analytical and constructive approaches. We extracted the most promising approaches based on our subjective judgment, which relied on the published industrial case studies and tool support, and summarized them in a guidelines document.

The guidelines document contains approx. 20 different approaches and is structured along the software development phases (e.g., requirements, architecture, implementation), but implies no waterfall model. The phases merely improve the orientation of the reader and are aligned with internal ABB guidelines for software engineering.

Our method descriptions focus on the relevance for sustainability and motivate the use of a method from the perspective of a developer or architect who wants to tackle an evolution scenario. We additionally coarsely estimated the learning and application effort of each method and provided references to respective tools. In order to focus on sustainability guidelines without providing generic software engineering best practices, we conducted several stakeholder interviews.

In the following, we briefly summarize selected methods and tools for each phase and discuss some of their risks (Fig. 3).

Sustainable Requirements: Documenting, prioritizing, analyzing, and tracing functional and non-functional requirements to an industrial software system is an important prerequisite for sustainability. Due to the complexity and longevity of industrial systems, tool support (e.g., DOORS, CaliberRM, HP Quality Center) is essential. Well managed requirements preserve the knowledge about the design rationale of a system and ease decision tracking. Requirements tracing is helpful for long-term development with changing personal as it preserves the knowledge about why certain design decisions were made or why certain parts of the code exist. Additionally, each requirement should be analysed for its potential impact on sustainability as early as possible to be able to improve the upfront

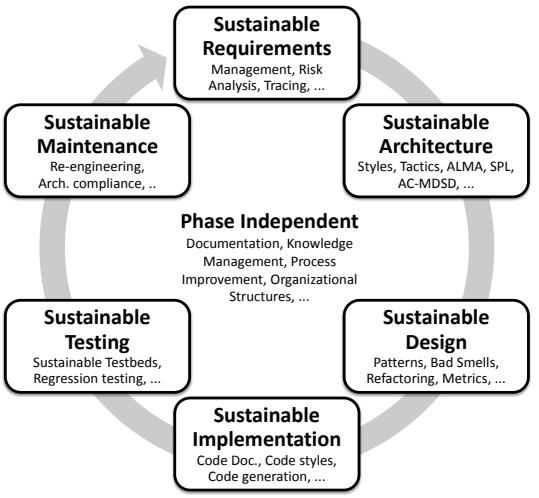


Figure 1: Structure of Software Sustainability Guidelines

design of a system.

Several *risks* are associated with sustainable requirements capturing. Requirements can be incomplete, contracting, or unstructured. Their impact on system sustainability is often not systematically analysed. Many future requirements during system evolution cannot be foreseen. Trade-offs between maintainability requirements and requirements for other quality attributes (e.g., performance, reliability, safety) require a thorough analysis. Especially for long-living systems, there is a danger that the requirements themselves are not maintained and updated during the evolution of the system, which then complicates further evolution.

Sustainable Architecture: Several heterogeneous approaches are known for designing sustainable software architectures. Scenario-based architecture evaluation methods (e.g., ATAM [CKK02], ALMA [BLBvV04]) can be used to elicit evolution scenarios from various stakeholders and to discuss their impacts on software components and reveal potential ripple effects through the architecture. Change-oriented architectural styles and patterns (e.g., layering, pipe-and-filter, plug-ins) and architectural modifiability tactics (e.g., localizing changes, deferring binding time [BCK03]) help to prepare the architectural design for evolution. Even more profound, software product lines help to manage system variability, while architecture-centric model-driven development is a potential mean to cope with platform adaption.

Risks associated to sustainable software architecture design are for example undocumented design rationale or too much flexibility. Missing design rationale complicates the evolution of a system and can potentially be mitigated through architecture knowledge management [BDLvV09]. While too few flexibility in the architecture limits evolution options, too much flexibility increases architectural complexity and thus maintainability (i.e., understandability, analyseability, modifiability).

Sustainable Design: Methods for improving the design (i.e., more low level structures,

code-related) of an industrial software system towards sustainability are the application of design patterns, refactoring, and code metric monitoring. Many design patterns [GHJV95] increase the flexibility and extensibility of code (e.g., abstract factory, bridge, decorator) and thus improve sustainability on a small scale. Bad smells are critical code areas which might incure high maintenance costs. They can be mitigated semi-automatically through refactoring tools [FBBO99]. To detect undesired structures in the code there are several tools (e.g., Findbugs, SISSy). Further tools (e.g., ISIS) monitor and aggregate certain code and development metrics. While these methods are largely standard software engineering best practice, they need organizational support.

Design *risks* are associated with creating an untouchable design, wrong abstraction level, or misleading metrics. Design principles should not be changed on a daily basis, but still be regularly assessed and adapted where appropriate. Too low design abstraction levels require substantial efforts during evolution, while too high abstraction levels might miss some important information. Design metrics and laws are not fixed and can require deviations in rare cases, which should be documented. Small-scale sustainability requires a special design and regularly planned refactorings.

Sustainable Implementation: During implementation, the sustainability of a system can be increased by writing clean code, documenting code, and/or code generation. Clean code adheres to specific coding guidelines, assists understandability, and thus lowers the effort for maintenance tasks. In the same manner, code documentation and naming conventions can improve understandability. Tools for codestyle conformance checks are for example Checkstyle or FxCop. Executable UML has found initial application in the industry and the corresponding code generation may lead to saved implementation efforts and higher code quality thus increasing sustainability.

Some implementation-related *risks* are poor code quality, final version thinking, and the use of unproven technologies. Furthermore, due to the longevity of industrial systems, their initial development environment tools may get lost, which then complicates further changes.

Sustainable Testing: While software testing is important for overall software quality, for sustainable systems continuous integration testing, regression testing, and sustainable testbeds have special importance. Continuous integration testing implements continuous processes of applying quality control. Regression testing checks whether new bugs are introduced during maintenance activities or while fixing bugs. Reusable unit tests facilitate change, simplify integration, and serve as way of documentation. A sustainable test-bed can for example rely on virtualization techniques to emulate older hardware, operating systems, and development environments.

Testing *risks* are associated with practically limited testing coverage, costs for testing, and the evolution of tests. Formal verification might be effective for some smaller safety-critical systems, but requires checking the underlying verification assumptions.

Sustainable Maintenance: Many typical maintenance activities (e.g., refactoring, checking code-metrics, applying patterns) are also part of the design and implementation phase. Hence, these activities are not repeated in this paragraph. Additional maintenance techniques relevant for sustainability and applied in industry are architecture consistency

checking and reverse engineering. Tools for checking the architectural consistency (e.g., Lattix, ndepend) can help to avoid architecture erosion, i.e., the situation that the implementation violates architectural constraints (e.g., strict layering between subsystems). Reverse engineering is applied on legacy systems, for example when no architectural design documentation is available. Reverse engineered higher-level structures improve understandability and may allow to estimate the impact of system modifications.

Risks associated with maintenance are for example architecture erosion and a lost overview. Architecture erosion might lead to a violation of a system's initial requirements and design principles and is thus undesirable. On the other hand, the implementation should not become a slave to the architecture, which might get outdated due to a changing environment or stakeholder requirements. Without a complete system overview, maintenance activities with a narrow focus can further decrease maintainability.

Sustainable Phase-independent Methods: Some approaches are cross-cutting and concern all development phases. These include for example process improvements, organizational support, sustainable documentation as well as knowledge management. Sufficient system documentation can prevent the loss of knowledge in a long-living system due to a change of personal. Many empirical studies have found a positive influence on system maintenance and evolution activities due to good documentation. Examples for documentation tools are JavaDoc, Doxygen, and Wikis. Design rationale can be handled through architecture knowledge management tools [TAJ+10] and thus supports traceability and maintenance activities.

4 Validation

We have not yet performed an extensive validation of our sustainability guidelines against the described evolution scenarios. However, this section discusses our validation goals (Section 4.1), maps one selected evolution scenario to the guidelines (Section 4.2), and sketches planned case studies (Section 4.3).

4.1 Validation Goals

The validation of our sustainability guidelines (i.e., evidence of benefits both technically and financial) is an essentially complex endeavor. First, the usability of the guidelines document itself needs to be validated. Second, the proposed methods must be validated through application and collection of experience.

To validate the *usability* of the guidelines, we propose a qualitative study first among a selected number of ABB architects and developers and then also among other developers from the industrial automation domain. Developers should be asked, whether the guidelines reflect the necessary amount of information to decide on the application of a method and whether the guidelines can be mapped to their concrete problems. As an initial step in this direction, we provide a mapping of a concrete evolution scenario to the guidelines in

Section 4.2.

Validating the *applicability* and usefulness of the recommended methods themselves is much more complicated than assessing the usability of the guidelines document. Replicated empirical studies would be necessary for the recommended methods to get trustworthy results. Validation criteria are whether evolution costs can be reduced (e.g., fewer modification necessary), whether software quality increases (e.g, lower number of failure reports), and whether maintenance tasks can be carried out faster (e.g., lower time to fix a bug). A complete validation is out of scope for our research in the short term. However, it is possible to refer to literature for already documented industrial case studies, and furthermore, we plan a number of smaller case studies to initially assess some selected methods (Section 4.3).

Empirical studies and experience reports about selected sustainability methods can be found in literature. For example, Sommerville et al. [SR05] have evaluated a maturity model for *requirements* engineering and tried to determine the impact on the quality of the resulting software product. In the area of software *architecture*, there are for example more than 20 industrial case studies reported for ATAM [CKK02] and 7 case studies for ALMA [BLBvV04]. Sustainability guidelines for software *design* can rely on empirical studies on refactoring [MT05] and the impact of applying design patterns [PULPT02]. Graves et al. [GHK+01] analysed the impact of different strategies for regression *testing*.

4.2 Mapping the Guidelines to a Sample Scenario: Third party components

The integration of third party components into automation systems is a typical evolution scenario. When looking at the different life- and evolution-cycles of automation systems and third party components, developers have to ensure that the integration is done in a way that future independent changes are easily possible without costly rework. For example, middleware frameworks evolve significantly over the lifetime of a process automation system, e.g. Java EE had 5 major revisions since its first release in 1999.

In order to ease typical development tasks, we ranked the relevance of recommended activities in our guidelines for every evolution scenario. In this section we briefly show this ranking for the evolution scenario of *integrating a third party component* and map it to a concrete development task. Only those phases and activities with high relevance are included.

Requirements Activity / Guideline: Management / Document why the 3rd party component is selected and which alternatives exist.

Mapping: Based on the performance and connectivity requirements of the automation system, a communication middleware was selected after a thorough technology evaluation.

Activity / Guideline: Tracing / Capture where the 3rd party component is assembled and deployed.

Mapping: The used capabilities of the middleware component are traced from requirements to system components down to implementation artifacts.

Architecture Activity / Guideline: Patterns, Styles, Tactics / Clean encapsulation of the component; if uncertainties are implied by 3rd party component: make it a variation point by use of a pattern (e.g. Factory).

Mapping: The usage of the middleware has been isolated in dedicated communication components, e.g. connect components for realizing connectivity through standardized automation protocols.

Design Activity / Guideline: OO-Metrics / Check encapsulation of 3rd party components; strict interface communication.

Mapping: A code inspection tool is used to monitor usage of public API calls of the middleware.

Testing Activity / Guideline: Unit Testing, Integration / Test API, test system integration with 3rd party components.

Mapping: Conformance tests for the middleware are used to ensure its quality; unit tests are in place to find functional regressions, and system tests find quality differences, e.g. performance degradation.

Maintenance Activity / Guideline: Architecture consistency checking / Check strong dependencies to component and existence of clear interfaces.

Mapping: Advanced static analysis tools (Lattix, Klocwork) are used to enforce conformance to the system architecture. The automation system has to be portable to several operating systems, thus the usage of the middleware has been restricted to isolated components, which is checked as well.

4.3 Planned Case Studies

While the former mapping provides an initial validation of the usefulness of our sustainability guidelines, we plan further case studies to evaluate a small number of the recommended methods. We identified three evolution scenarios from ABB products, which shall be assessed and analysed post-mortem.

The first case study shall apply the ALMA method [BLBvV04] to compare two software architecture designs of a controller firmware for their support of sustainability. The second case study analyses the evolution problems of a third party component in a process control system using the SISSy code analysis tool searching for bad smells and sustainability impeding hot spots. The third case study shall redocument architectural design rationale using a tool for software architecture knowledge management [BDLvV09].

5 Related Work

Several surveys in the area of software evolution can be found in literature. Chapin et al. [CHK+01] propose a redefinition of the types of software evolution and software

maintenance. A classification of software evolution types is presented by distinguishing changes of (1) the software, (2) the documentation, (3) the properties of the software, and (4) the customer-experienced functionality.

Godfrey and Buckley et al. [BMZ+05] create a taxonomy of software change based on characterizing the mechanisms of change and the factors that influence these mechanisms. The ultimate goal of this taxonomy is to provide a framework that positions concrete tools, formalisms and methods within the domain of software evolution.

Mens et al. surveyed software refactoring [MT05] and provided a list of challenges in software evolution [MWD+05]. Germain [GG08] discuss the past, present, and future of software evolution. They provide definitions of evolution and maintenance terminology as well as a comparison with biological evolution.

Rozanksi and Woods [RW05] define an "evolution perspective" in their architectural framework "viewpoints and perspectives". They discuss how architectural changes can be described and characterized and list some abstract evolution tactics, but exclusively focus on the software architecture. Bass et al. [BCK03] list a number of modifiability tactics for software architectures. They focus on modifiability and do not include requirements or implementation related issues.

6 Conclusions

This paper addresses the problem of developing sustainable long-living software systems in the industrial automation domain. Based on our experience in this domain, we provided a list of generic evolution scenarios, which can be used in future evolution evaluation studies. Additionally, we have performed an extensive literature search and extracted a number of sustainability guidelines for different development phases. We initially sketched the validation of these guidelines in this paper.

Our sustainability guidelines shall mainly help practitioners in the industrial automation domain to develop software with higher quality and lower evolution costs. The guidelines provide an overview and evaluation of selected, matured approaches, and a list of associated risks. Researchers can also rely on these guidelines and identify potential issues for future research.

In our future work, we will extend and improve the guidelines and apply them to three small case studies to understand the value of a selected number of the recommended methods.

References

[BCK03] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[BDLvV09] Muhammad Ali Babar, Torgeir Dingsyr, Patricia Lago, and Hans van Vliet, editors. *Software Architecture Knowledge Management: Theory and Practice*. Springer, 1st

edition, August 2009.

[BG09] M.A. Babar and I. Gorton. Software architecture review: The state of practice. *Computer*, 42(7):26–32, 2009.

[BLBvV04] P.O. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet. Architecture-level modifiability analysis (ALMA). *Journal of Systems and Software*, 69(1-2):129–147, 2004.

[BMZ+05] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel. Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(5):309–332, 2005.

[CHK+01] N. Chapin, J.E. Hale, K.M. Khan, J.F. Ramil, and W.G. Tan. Types of software evolution and software maintenance. *Journal of software maintenance and evolution: Research and Practice*, 13(1):3–30, 2001.

[CKK02] P. Clements, R. Kazman, and M. Klein. *Evaluating software architectures: methods and case studies*. Addison-Wesley Reading, MA, 2002.

[DKK+11] Zoya Durdik, Heiko Koziolek, Klaus Krogmann, Johannes Stammel, and Roland Weiss. Software Evolution for Industrial Automation Systems: Literature Overview. Technical Report 2011-2, Faculty of Informatics, Karlsruhe Institute of Technology (KIT), January 2011. ISSN: 2190-4782.

[FBBO99] Martin Fowler, Kent Beck, John Brant, and William Opdyke. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman, 1999.

[GG08] M.W. Godfrey and D.M. German. The past, present, and future of software evolution. In *Frontiers of Software Maintenance, 2008. FoSM 2008.*, pages 129–138. IEEE, 2008.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*, volume 206. Addison-wesley Reading, MA, 1995.

[GHK+01] T.L. Graves, M.J. Harrold, J.M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 10(2):184–208, 2001.

[KSB10] Heiko Koziolek, Bastian Schlich, and Carlos Bilich. A Large-Scale Industrial Case Study on Architecture-based Software Reliability Analysis. In *Proc. 21st IEEE International Symposium on Software Reliability Engineering (ISSRE'10)*, pages 279–288. IEEE Computer Society, November 2010.

[MT05] T. Mens and T. Tourwé. A survey of software refactoring. *Software Engineering, IEEE Transactions on*, 30(2):126–139, 2005.

[MWD+05] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. Challenges in software evolution. In *8th Int. Workshop on Principles of Software Evolution (IWPSE'2005)*, pages 13–22. IEEE, 2005.

[PULPT02] L. Prechelt, B. Unger-Lamprecht, M. Philippsen, and WF Tichy. Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. *Software Engineering, IEEE Transactions on*, 28(6):595–606, 2002.

[RW05] N. Rozanski and E. Woods. *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley Professional, 2005.

[SR05] I. Sommerville and J. Ransom. An empirical study of industrial requirements engineering process assessment and improvement. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 14(1):85–117, 2005.

[TAJ+10] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. Ali Babar. A comparative study of architecture knowledge management tools. *Journal of Systems and Software*, 83(3):352–370, 2010.

Introducing Adaptivity to Achieve Longevity for Software

Mahdi Derakhshanmanesh[1], Jürgen Ebert[1], Mehdi Amoui[2], Ladan Tahvildari[2]
[1]University of Koblenz-Landau, Germany, Institute for Software Technology
[2]University of Waterloo, Canada, Software Technologies Applied Research
Email: {manesh|ebert}@uni-koblenz.de, {mamouika|ltahvild}@uwaterloo.ca

Abstract: *Long living software systems* (LLSSs) must provide the flexibility to react to changes in their operating environment as well as to changes in the user's requirements, even during operation. *Self-adaptive software systems* (SASSs) face adaptivity at runtime within predefined bounds. Yet, not all types of necessary variations can be anticipated and unforeseen changes to software may happen. Thus, systems that are meant to live in such an open-ended world must provide self-adaptivity (*micro adaptation*), but there is an additional need for adaptability of the system so that it can be adjusted externally (*macro adaptation*). This paper gives an overview of the *graph-based runtime adaptation framework* (GRAF) and sketches how it targets both types of adaptation.

1 Introduction

As the lifetime of software tends to get even longer in the future, such *long living software systems* (LLSSs) must be adaptable to changing requirements. These requirements may stem from varying user needs or from changes in the software's operating environment. Ideally, an LLSS is capable of reacting to such changes during operation, without showing any noticeable down-time in service to the system's users. But, in practice there have to be adaptive maintenance actions as well.

A *self-adaptive software system* (SASS) is designed to face *foreseen* changes in its operating environment. On such a change, an SASS is still capable of fulfilling its set of requirements. More generally, an SASS is a *reflective* system that is able to modify its own behavior in response to changes in its operating environment [OGT+99]. It may also be called *autonomic* and/or *self-managing* system [ST09].

Although SASSs today can already deal with changes in their environment, they are not well-suited to tackle an *open-ended world*, which is especially characterized by *unforeseen* changes [SEGL10]. To handle these changes, classical approaches for software evolution have to be applied.

In this paper, we present the *graph-based runtime adaptation framework* (GRAF) [DAET], which supports the introduction of variability to software and goes beyond the goals of achieving runtime adaptivity for a predefined problem space only.

By explicitly making self-adaptation of the SASS at runtime (*micro adaptation)* as well as maintenance tasks performed by an external subject (*macro adaptation*) two comple-

mentary parts of an integrated process, the introduction of runtime adaptivity to software can help with incorporating the required flexibility for longevity of software. From that perspective, we see adaptivity as the main property of software to be adjustable to any changes in the environment, no matter if adaptation is actually performed online (on its own or with external help) or offline.

In our CSMR 2011 short paper [ADET11], we propose a flexible, model-centric architecture for SASSs that supports a process for migrating non-adaptive software towards self-adaptive software. In a second paper [DAET], we present the details of GRAF and its abilities in the area of micro adaptation together with a comprehensive case study.

This paper complements the preceding publications by focusing on the use of GRAF for achieving longevity of software. It is organized as follows. In Section 2, we describe the GRAF framework, which implements a flexible, model-centric architecture for SASSs, as proposed in [ADET11]. Then, we introduce the core ideas behind *micro* and *macro* adaptation in Section 3 and give an overview on the implementation work and the case studies done up to now in Section 4. We briefly cover related work in Section 5. Finally, we conclude this paper in Section 6 and give an outlook on possible areas of future work.

2 Architecture of GRAF

In the context of migrating existing, non-adaptive software towards an SASS, Amoui et al. present a model-centric architecture that is designed around an external adaptation framework in [ADET11].

The *graph-based runtime adaptation framework* (GRAF) acts as an *external controller* of the adaptable software. GRAF is not just an adaptation manager. It encapsulates a *runtime model* that is specific to the adaptable software. The framework achieves adaptivity of the adaptable software's behavior by *querying*, *transforming* and *interpreting* parts of the runtime model. The main layers and components are illustrated in Figure 1.

Subsequently, we discuss the structure of the adaptable software and GRAF as well as the connecting interfaces. The responsibilities and tasks of each framework layer are introduced and we explain how each of them contributes to the whole architecture of an SASS that can be extended in reaction to unforeseen changes.

2.1 Adaptable Software

In the context of GRAF, we refer to the software to be managed by the framework as the *adaptable software*. It is set up in a way so that it can be controlled by GRAF, which plays the role of the external adaptation management unit. Adaptable software can be built by migrating an existing, non-adaptable software system towards adaptability [ADET11]. Alternatively, it can be developed from scratch.

The adaptable software is *causally connected* to the runtime model which is encapsulated

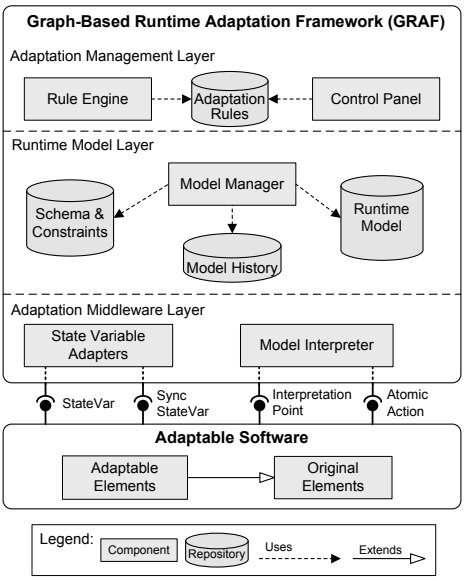


Figure 1: A SASS that is using GRAF to achieve adaptivity.

and managed by GRAF. This causal connection is established by an intermediate layer, the *adaptation middleware*, which communicates with the adaptable software using predefined interfaces.

When applying GRAF in a migration context, *original elements* are those software elements (e.g., classes or methods) that are already existing within the software application to be migrated towards a SASS. In the context of creating adaptable software from scratch, these elements can be thought of as helper elements, such as classes provided by imported libraries.

In a migration towards self-adaptive software, *adaptable elements* are derived from original elements by applying refactoring steps and by exposing them via predefined interfaces to GRAF. They are the building blocks that support a certain degree of variability by either providing data about changes, or, by offering actions to be used for adjusting the system.

Every adaptable software element needs to provide some of the interfaces for adaptation provided by GRAF as described in Table 1.

2.2 Graph-Based Runtime Adaptation Framework

The GRAF framework itself consists of three layers, namely the *adaptation middleware layer*, which establishes the connection to the adaptable software, the *runtime model layer*, which encapsulates and handles the runtime model and the *adaptation management layer*, which controls the adaptation actions at runtime either triggered by events stemming from

Table 1: Interfaces to be provided by any adaptable software for interaction with GRAF.

Interface	Description
`StateVar`	exposes variables that hold information about the operating environment. Changes in their values are propagated to the model manager, which decides if the value is (i) stored in the runtime model (*Reification*) or (ii) discarded.
`SyncStateVar`	exposes variables similar to `StateVar`, but their representation in the runtime model can be also changed from outside. The new value is then propagated back to the managed adaptable software.
`InterpretationPoint`	exposes handles to positions in the adaptable software's control flow (*interpretation points*) at which an associated part of the behavioral model shall be executed (by interpretation).
`AtomicAction`	exposes methods that can be used as *atomic actions* in the behavioral model. They are used by the interpreter to execute individual actions that are composed inside of the behavioral model.

the model layer or by external actions. These layers are described in more detail in the following.

2.2.1 Adaptation Middleware Layer

The *adaptation middleware layer* is stateless and responsible for all possible ways of communication and interaction between the adaptable software and the runtime model layer. It provides a set of reusable components that are independent from the adaptable software.

State Variable Adapters. *State variable adapters* support the propagation of (changed) values from annotated variables in the adaptable elements to the runtime model by using the `StateVar` interface. In addition, the `SyncStateVar` interface also uses these adapters when propagating tuned variable values from their runtime model representation back into the adaptable software.

Model Interpreter. For adaptivity of behavior, GRAF's *model interpreter* executes parts of the behavioral model that compose *actions* available via the `AtomicAction` interface. That way, behavior for a specific point in the adaptable application's control flow (*interpretation point*) within adaptable elements is based on a model. Interpretation points are the starting points of methods that need adaptivity and, hence, are exposed to the interpreter

by the `InterpretationPoint` interface.

The model interpreter is specific to a given runtime model schema. We give an excerpt in Figure 2. Each `Action` in the behavioral model is associated with an existing, implemented method in the adaptable software. Starting at an `InitialNode` and walking along a path in the behavioral model, the interpreter resolves conditions (expressed as queries on the runtime model) at `DecisionNode` vertices. Methods of the adaptable software are invoked via the `AtomicAction` interface and using Java reflection for each `OpaqueAction`. Interpretation terminates when a `FinalNode` vertex is reached and the adaptable software continues executing.

2.2.2 Runtime Model Layer

The *runtime model layer* contains the runtime model that is at the core of the GRAF approach to achieve adaptivity. In addition, this stateful layer contains utility components that simplify and encapsulate necessary operations on the graph-based runtime model, such as evaluating queries or executing transformations.

Runtime Model. The design of GRAF's architecture is centered around a *runtime model*. Essentially, this model needs to (i) offer a view on the adaptable software's inner state as well as (ii) describe behavior that can be executed by an interpreter. Hence, the runtime model serves as a view on the adaptable software and (iii) is controllable and modifiable by the rule engine.

In GRAF, the runtime model conforms to a customizable *schema* (meta-model) that is described as UML class diagrams. GRAF automatically generates Java source code to create and manipulate models that conform to the schema as described in UML.

An excerpt of the schema that is currently used is illustrated in Figure 2. According to this schema, every runtime model contains a set of `StateVariable` vertices to store exposed data from the adaptable software and `Activity` vertices to store entry points to behavior executable by runtime interpretation.

In this schema, behavioral models are kept as variants of UML activity diagrams which are represented by their *abstract syntax graphs*. In general, also other types of behavioral models such as Petri nets, statecharts, etc. could be used.

Schema and Constraints. GRAF supports the validation of *constraints* on the runtime model. A set of constraints can be derived from the runtime model schema. Examples are the multiplicities for associations that restrict the allowed edges between vertices in the runtime model's internal graph representation. To express further restrictions on the modeling language, complex constraints can be defined on the runtime model schema by using a query language.

Additionally, developers of a SASS can write constraints that are specific to the domain of their adaptable software and belong to the SASS's runtime model. For instance, it is possible to express that, for a certain behavioral model (activity), specific actions (methods)

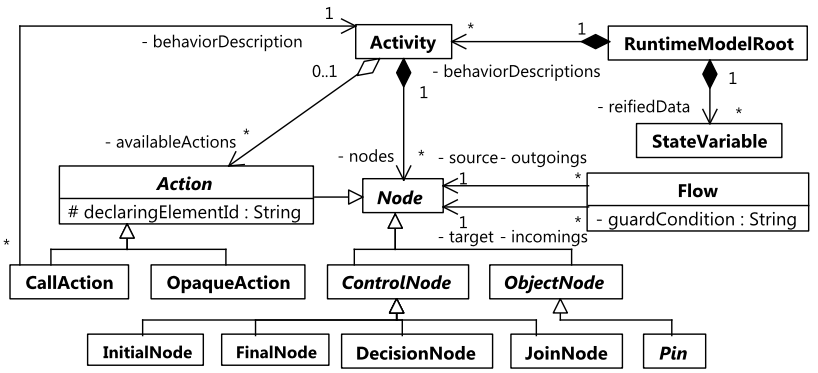


Figure 2: Excerpt of the runtime model schema.

are not allowed to be called in sequence. At present, constraint checking mechanisms have been implemented only prototypically and are not yet used in the adaptation loop.

Model Manager. The *model manager* is responsible for all tasks that are related to manipulating the runtime model. It keeps the runtime model in sync with the inner state of the adaptable software, which is exposed via the `StateVar` and `SyncStateVar` interfaces.

In addition, the model manager acts as a controller for all other types of accesses to the runtime model and hence, it provides a set of utility services for *querying* and *transforming*. Furthermore, the model manager is responsible for the evaluation of constraints after a transformation has been applied. It can even roll back changes to the runtime model and it informs the rule engine about this failed adaptation attempt.

Model History. The *model history* supports the storage of temporal snapshots of the runtime model as additional information (history/prediction). The rule engine can access this data by querying the history repository via the model manager. That way, the rule engine can learn from its past actions by using the available history. The application of data mining techniques such as *frequent itemset analysis* [AS94] are possible once the repository contains a representative amount of data. Moreover, the collected data may provide valuable information for maintaining the SASS. The model history has not been implemented yet.

2.2.3 Adaptation Management Layer

The *adaptation management layer* acts as the controller that *runs in parallel* to the adaptable software and uses a set of adaptation rules to make changes to the runtime model

over time. This stateful layer is composed of a repository with adaptation rules and a rule engine that uses its own heuristics to plan adaptation.

Rule Engine. In GRAF, the *rule engine* plays the role of an adaptation manager. It gets notified by events, for instance, whenever the runtime model changes. Moreover, it can receive additional information on the current or past state of the adaptable software by using the model history. After gathering all the required information for planning, the rule engine uses the set of available adaptation rules to choose a (compound) transformation to be executed on the runtime model via the model manager.

Adaptation Rules. Adaptation rules are *event-condition-action* rules that describe *atomic adaptation tasks*. The three parts of an adaptation rule are as follows:

1. **Event.** The event for the application of a rule occurs whenever the runtime model changes and the rule engine needs to react. The model manager keeps track of these events.
2. **Condition.** The condition is expressed by a boolean query on the runtime model that tests whether an adaptation action may be applied or not.
3. **Action.** The action is a model transformation and encodes the actual adaptation to be performed on the runtime model.

Control Panel. The *control panel* allows administrators to interact with the SASS and especially with GRAF. They can (i) observe the log of adaptation steps by querying the runtime model history, (ii) modify adaptation rules and model constraints as well as (iii) override decisions and heuristics of the rule engine. Furthermore, administrators get notified whenever an adaptation attempt fails or if the SASS is totally uncontrollable by the adaptation management layer. The presented version of GRAF does not implement the control panel.

3 Runtime Adaptivity and Evolution

If any runtime changes to the SASS that are carried out autonomously by itself (*micro adaptation*) are not sufficient, some of its components or repositories must be maintained externally and possibly manually (*macro adaptation*). Due to the limits of SASSs in dealing with unforeseen situations as well as the increasing entropy (disorder) that may be caused by self-adaptation, phases of micro and macro adaptation need to alternate to achieve longevity. In this section, we exemplarily discuss micro and macro adaptation, assuming that the SASS has been created using GRAF.

3.1 Micro Adaptation

Subsequently, we give an overview of the main steps during micro adaptation, disregarding all the possible alternate flows and exceptions that might occur on purpose here. A detailed description is given in [Der10].

Assuming application-level adaptation, as discussed in [ST09], these changes lead to changes of the *inner state* of the adaptable software (the *self*) and thus, can be sensed in its variables and fields. Changed values from the adaptable software's inner state are exposed to state variable adapters via the `StateVar` interface and finally propagated to the runtime model. This results in a notification of the rule engine in the adaptation management layer.

Following that, GRAF executes steps similar to the *MAPE-loop* as described in IBM's architectural blueprint for autonomic computing [IBM06]. When a change listener of the rule engine receives an event from the model manager, the rule engine analyzes the change and plans a (compound) transformation to be applied on the runtime model via the model manager. Afterwards, the runtime model is adjusted.

Finally, an actual behavior change of the managed adaptable software has to be achieved. Two different ways are supported in GRAF: (i) adaptation via state variable values and (ii) adaptation via interpretation of a behavioral (sub-)model.

In the first case, state variables can be chosen so that changes to them are reflected back to the adaptable software. For this mechanism to work, the corresponding variable of the adaptable application must be exposed to GRAF via the `SyncStateVar` interface. If the runtime model value of such a state variable is changed by a transformation, the adaptable software is made aware of the new value whenever the variable is *used* in the adaptable software.

In the second case, GRAF supports the modeling of behavior (using UML activity diagrams at the moment). During execution of some operation of the adaptable software which is exposed to GRAF via the `InterpretationPoint` interface, the model interpreter is invoked and finds the associated behavioral model. If this behavior is default behavior, there is no need for adaptation and the interpreter returns this information so that the adaptable element executes the existing default code block in the adaptable software. Otherwise, the model interpreter executes the associated part of the behavioral model using the `AtomicAction` interface, replacing a part of the adaptable software's default control flow.

3.2 Macro Adaptation

In cases where the SASS faces new challenges that it cannot tackle adequately, the system must be evolved, that is, an external subject (here, the human administrator) must be able to adjust the system. Due to the flexibility of the proposed architecture, many of these maintenance tasks can be performed at runtime. Still, there are cases where the

SASS must be stopped and restarted again when technical reasons require it, e.g., after modifications to the source code. In this section, we discuss ideas and sketch macro adaptation in the context of SASSs built around GRAF. We start top-down, from the adaptation management layer.

Ways to extend the adaptivity property of the SASS are to (i) *add* new adaptation rules to the repository, (ii) *modify* existing adaption rules in terms of their queries and transformations or even (iii) *remove* adaptation rules. Such changes can be made via the external control panel. Ideally, the rule engine does not have to be adjusted. Changes to its analyzing and planning algorithms may be needed, though. By separating them into different modules and keeping them independent from the core, this issue can be solved.

At the runtime model layer, further adjustments are possible. Among the smaller changes, there are modifications similar to the adaptation rules (add, modify, remove) that can be performed for application-specific constraints. Likewise, changes to the runtime model may be desirable, such as refining parts of the behavioral model. More complex changes are schema-related adjustments. Developers may want to change the language for modeling the runtime model and use Petri nets or statecharts instead of UML activity diagrams. Given such a heavy change, the adaptation rules will be affected as well and the model interpreter has to be adjusted or even replaced.

As the adaptation middleware is a generic communication layer, it stays stable for most of the time. Only a change of the runtime model schema (description of syntax) will result in a re-write of the model interpreter (description of semantics). Changes to state variable adapters are only needed in cases where the concept of state variables is modeled differently by the runtime model schema.

There might be necessary changes at the level of the adaptable software as well. In cases where existing state variables do not provide data about changes sufficiently, new ones have to be introduced and exposed to GRAF. Again, existing state variables can be deleted as they may become obsolete over time. Furthermore, the existing atomic actions may not provide enough expressiveness to model the desired behavior and the pool of available, exposed actions needs to be refined.

The necessity to start a macro adaption should be detected by the framework itself. This can be done by special rules that test some quality properties of the model or by analyzing the history information periodically, for instance.

Finally, *traceability* between different elements at all layers is important, e.g., as atomic actions are removed, they may no longer be used in the behavioral model and adaptation rules shall not create variations of behavior that make use of these missing actions.

4 Implementation and Case Studies

GRAF is implemented around a small set of interfaces and abstract classes in Java and each of its components can be extended or replaced with minor effort, thus, making it suitable for constructing long living, self-adaptive Java software systems.

Java *annotations* provide an easy way to add meta-data to existing source code without altering its functional properties. With the help of *aspect oriented programming* (AOP) techniques and a powerful AOP library like JBoss AOP[1], these annotations are then used in pointcut expressions resulting in automatic instrumentation of the byte-code to connect the adaptable software to GRAF via its middleware.

GRAF's runtime model and its operations are based on graph technology. Models, schemas as well as queries and transformations are implemented in the technological space of *TGraphs* [ERW08]. Using the *IBM Rational Software Architect* [2](RSA), the runtime model schema is developed by drawing UML class diagrams. Based on the runtime model schema, a Java API is generated using JGraLab[3]. The generated Java classes represent vertex- and edge-types of runtime models (TGraphs) that conform to the runtime model schema. The generated API is then used for creating, accessing, and modifying runtime models. While transformations, the action part of adaptation rules, are currently realized using the generated Java API, we use the *Graph Repository Query Language* (GReQL) for checking constraints and retrieving data from the runtime model TGraph.

We successfully applied GRAF to the open-source VoIP application OpenJSIP [4], where adaptivity was introduced to achieve less timeouts and a more clear reaction of the system to the user under high system load [Der10]. Furthermore, we performed a more comprehensive case study with the game engine Jake2 [5], where the SASS adapts the behavior of artificial players depending on the human player's skills [DAET]. An increase in used memory (model representation, JBoss AOP wrappers) and in execution times (model interpretation) is measurable in both case studies, but this did not affect the user's experience.

5 Related Work

Schmid et al. motivate in [SEGL10] that as future systems will operate even longer, they must be capable of coping with changes in their environment or their requirements during operation. Although self-adaptive software can handle such challenges within given bounds, *long living software systems* need to be extendable and adaptable in an open-ended world, where changes are unforeseen. Schmid et al. propose the modularization of models for evolving adaptable systems and discuss an adaptive system with adaptivity enhancement. This paper strengthened our motivation to present GRAF in the context of longevity.

The DiVA project [MBJ+09] considers design and runtime phases of adaptation. It is based on a solid four dimensions modeling approach to SASS [FS09]. At design time, a base model and its variant architecture models are created. The models include invariants and constraints that can be used to validate adaptation rules. In comparison to our GRAF approach, DiVA also focuses on adaptation at the architectural level, whereas we target

[1] `http://www.jboss.org/jbossaop`
[2] `http://www-01.ibm.com/software/awdtools/swarchitect/websphere/`
[3] `http://www.ohloh.net/p/jgralab`
[4] `http://code.google.com/p/openjsip/` (Last access: 14^{th} of January, 2011)
[5] `http://bytonic.de/html/jake2.html` (Last access: 14^{th} of January, 2011)

adaptation at a lower level by incorporating methods and fields as well.

Vogel and Giese [VG10] propose a model-driven approach which provides multiple architectural runtime models at different abstraction levels. They derive abstract models from the source models. Subsets of the target models focus on a specific adaptation concern to simplify autonomic managers. Similar to DiVA, this research also targets adaptivity from an architectural point of view.

The Rainbow framework [CHG+04] uses an abstract architectural model to monitor the runtime properties of an executing system. Similar to GRAF, certain components of Rainbow are reusable and the framework is able to perform model constraint evaluation. However, this framework is mainly targeting architecture-level adaptation, where adaptive behavior is achieved by composing components and connectors instead of interpreting a behavioral model.

Similar to some of the presented work, our implementation of GRAF makes use of *aspect oriented programming* for binding the adaptable software to the framework. In contrast to the mentioned research projects, we focus on the *interpretation of behavioral models* for achieving adaptability at runtime, while also supporting parameter adaptation.

6 Conclusions and Future Work

In this paper, we presented an approach for achieving runtime adaptivity in software based on the model-centric adaptation framework GRAF. Following the proposal for a model-centric SASS in [ADET11], we implemented GRAF to support the migration of existing software towards a SASS. The construction of an SASS from scratch is supported as well.

This framework is based on a layered architecture that features (i) a *middleware layer* for managing the coupling and communication of GRAF with the adaptable software, (ii) a *runtime model layer* that reflects the current state of the adaptable software and its behavior at certain parts that need adaptivity as well as (iii) an *adaptation management layer* with a rule engine that is able to adapt the runtime model by using on given adaptation rules. A first case study is described in [Der10] and a more comprehensive one will be available in [DAET].

Based on the view of micro and macro adaptation, we plan to experiment with different scenarios to learn more about sequences of alternation between these two phases. Finally, we see additional future work in the area of designing an evolution process that combines (i) the knowledge about *traditional maintenance* with (ii) the flexibility of *runtime adaptivity* given by self-adaptive software systems. We believe, that integrating these two areas provides a promising way towards achieving longevity in software.

Acknowledgement. Our thanks go to the anonymous reviewers for their thorough and inspiring feedback.

References

[ADET11] Mehdi Amoui, Mahdi Derakhshanmanesh, Jürgen Ebert, and Ladan Tahvildari. Software Evolution Towards Model-Centric Runtime Adaptivity. In *Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, Oldenburg, Germany, 2011. To appear.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[CHG+04] Shang-wen Cheng, An-cheng Huang, David Garlan, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37:46–54, 2004.

[DAET] Mahdi Derakhshanmanesh, Mehdi Amoui, Jürgen Ebert, and Ladan Tahvildari. GRAF: Graph-Based Runtime Adaptation Framework. Submitted elsewhere.

[Der10] Mahdi Derakhshanmanesh. Leveraging Model-Based Techniques for Runtime Adaptivity in Software Systems. Master's thesis, University of Koblenz-Landau, Germany, 2010.

[ERW08] Jürgen Ebert, Volker Riediger, and Andreas Winter. Graph Technology in Reverse Engineering, the TGraph Approach. In Rainer Gimnich, Uwe Kaiser, Jochen Quante, and Andreas Winter, editors, *10th Workshop Software Reengineering (WSR 2008)*, volume 126 of *GI Lecture Notes in Informatics*, pages 67–81, Bonn, 2008. GI.

[FS09] Franck Fleurey and Arnor Solberg. A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*, pages 606–621, 2009.

[IBM06] IBM. An Architectural Blueprint for Autonomic Computing. *Autonomic Computing*, 2006. White Paper.

[MBJ+09] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg. Models@ Run.time to Support Dynamic Adaptation. *Computer*, 42(10):44 –51, October 2009.

[OGT+99] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An Architecture-Based Approach to Self-Adaptive Software. *Intelligent Systems and their Applications, IEEE*, 14(3):54–62, 1999.

[SEGL10] Klaus Schmid, Holger Eichelberger, Ursula Goltz, and Malte Lochau. Evolving Adaptable Systems: Potential and Challenges. In *2. Workshop Design For Future 2010 - Langlebige Softwaresysteme (L2S2)*, Bad Honnef, Germany, 2010.

[ST09] Mazeiar Salehie and Ladan Tahvildari. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):1–42, 2009.

[VG10] Thomas Vogel and Holger Giese. Adaptation and abstract runtime models. In *Proceedings of the Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 39–48, 2010.



Zweiter Workshop zur Zukunft der Entwicklung softwareintensiver, eingebetteter Systeme (ENVISION2020)

www.sse.uni-due.de/
ENVISION2020
22. Feb. 2011, Karlsruhe
@SE|11 SOFTWARE ENGINEERING

ENVISION 2020: Zweiter Workshop zur Zukunft der Entwicklung softwareintensiver, eingebetteter Systeme

Kim Lauenroth, Mark Rzepka

paluno - The Ruhr Institute for Software Technology
Universität Duisburg-Essen
Gerlingstraße 16, 45127 Essen
{kim.lauenroth; mark.rzepka}@sse.uni-due.de

1 Zielsetzung des Workshops

Softwareintensive, eingebettete Systeme unterstützen den Menschen schon heute in vielen Bereichen des Lebens – sichtbar und unsichtbar. Beispielsweise verbessern sie im Automobil die Sicherheit, regulieren das Klima in Gebäuden oder steuern medizinische Geräte bis hin zu ganzen Industrieanlagen. Experten prognostizieren für die Zukunft eine rasante Zunahme softwareintensiver, eingebetteter Systeme.

Die Ausweitung des Funktionsumfangs und die zunehmende Vernetzung eingebetteter Systeme führen gleichzeitig zu einer rasanten Zunahme der Komplexität, die auch im Entwicklungsprozess Berücksichtigung finden muss. Existierende Vorgehensweisen und Methoden stoßen bereits unter den heutigen Rahmenbedingungen (z.B. Zeit- und Kostendruck) an ihre Grenzen. Existierende Ansätze und Methoden müssen aufgrund der wachsenden Herausforderungen in Frage gestellt und in Teilen neu konzipiert werden.

Der Workshop ENVISION2020 verfolgt das Ziel, die Entwicklung und Diskussion zukünftiger Ansätze, Vorgehensweisen und Methoden zur Entwicklung softwareintensiver, eingebetteter Systeme zu fördern. Ein besonderes Augenmerk gilt dabei modellbasierten Entwicklungsansätzen.

2 Workshop-Organisation



ENVISION2020 wurde organisiert durch die Innovationsallianz SPES2020, gefördert durch das BMBF, Förderkennzeichen 01IS08045. SPES2020 hat sich zum Ziel gesetzt, neue und innovative Ansätze und Methoden für die Entwicklung softwareintensiver eingebetteter Systeme zu erforschen und zu erproben.

Ausgehend von der Analyse verschiedener Anwendungsdomänen (bspw. Automotive, Avionik und Energie) soll ein zusammenhängender, domänenübergreifender Entwicklungsansatz entstehen. Dieser Ansatz soll anhand der Bedürfnisse der jeweiligen Anwendungsdomänen angepasst und in Zusammenarbeit mit den Industriepartnern erprobt werden. Siehe http://www.spes2020.de für weitere Informationen zu SPES2020.

Organisationskomitee:
- Manfred Broy, *TU München*
- Martin Deubler, *TU München*
- Kim Lauenroth, *Uni Duisburg-Essen*
- Klaus Pohl, *Uni Duisburg-Essen*
- Mark Rzepka, *Uni Duisburg-Essen*

Programmkomitee:
- Christian Allmann, *Audi*
- Klaus Beetz, *Siemens*
- Ottmar Bender, *EADS*
- Stefan Jähnichen, *FIRST*
- Peter Liggesmeyer, *IESE*
- Ulrich Löwen, *Siemens*
- Ernst Sikora, *Audi*
- Matthias Weber, *Carmeq*

Darüber hinaus waren Marcus Mews und Mark-Oliver Reiser als externe Gutachter tätig.

3 Teilnehmer

Die 22 Teilnehmer des Workshops stammten zu etwa gleichen Teilen aus Forschung und Industrie. Unter den Teilnehmern aus der Industrie befanden sich Vertreter verschiedener Anwendungsgebiete, darunter Automatisierung, Automobilbau und Luftfahrt.

In der Vorstellungsrunde drückten die Teilnehmer insbesondere Interesse an aktuellen Trends aus und wünschten sich Einblicke in aktuelle Methoden zu erhalten sowie Anregungen für neue Entwicklungsansätze zu erhalten. Als Interessensschwerpunkte gaben die Teilnehmer die Gebiete Requirements Engineering, Architekturentwurf, modellbasierte Entwicklung und Testen, sowie Reengineering von eingebetteten Systemen an. Darüber hinaus wurde starkes Interesse an langlebigen eingebetteten Systemen geäußert.

4 Zusammenfassung der Workshopvorträge und -diskussionen

Im Rahmen des Workshops wurden fünf Beiträge vorgestellt und diskutiert. Des Weiteren fand eine offene Diskussionsrunde statt, bei der die Erwartungen an Embedded Systems im Jahre 2020 gemeinsam mit allen Workshopteilnehmern diskutiert und hinterfragt wurden.

Im Folgenden geben wir einen Überblick über die Beiträge des Workshops und fassen die Diskussionen zu den einzelnen Beiträgen kurz zusammen. Anschließend werden die wichtigsten besprochenen Themen der offenen Diskussionsrunde vorgestellt.

4.1 Beitrag 1: A Seamless Model-Based Development Process for Automotive Systems

Der erste Vortrag “A Seamless Model-Based Development Process for Automotive Systems“ gehalten durch Jörg Holtmann, befasste sich mit einem durchgängigen Ansatz zur Entwicklung von Software in eingebetteten Systemen im Automobilbereich. Die vorgestellte Vorgehensweise ist konform zu Automotive SPICE. Der Ansatz beruht auf SysML, AUTOSAR und Modelltransformationen.

In der Diskussion hinterfragten die Teilnehmer das Ziel einer vollständigen Durchgängigkeit und erörterten, dass wesentliche Herausforderungen in der Berücksichtigung von Anforderungsänderungen und der Wiederverwendung von Altteilsystemen liegen. Als weitere Herausforderungen wurden die verteilte Entwicklung an mehreren Unternehmensstandorten und die Synchronisation mit anderen Entwicklungsmodellen (z.B. in MATLAB/Simulink) angeführt. Darüber hinaus wurde die Frage diskutiert, ob der vorgeschlagene Ansatz auf andere Domänen übertragbar ist.

4.2 Beitrag 2: Timing Overhead Analysis for Fault Tolerance Mechanisms

Im zweiten Vortrag „Timing Overhead Analysis for Fault Tolerance Mechanisms” stellte Kai Höfig einen Ansatz zur Untersuchung der Laufzeiten von komponentenbasierten Systemen mit Fehlertoleranzmechanismen vor. Das Ziel der vorgestellten Arbeit ist die Analyse der (maximalen) Gesamtlaufzeiten für verschiedene Laufzeitfehler. Um dies zu erreichen, werden erweiterte System- und Safety-Modelle zur Auswertung von Fehlermodi und Zeitkonstanten eingesetzt. Dies ermöglicht zu prüfen, ob die eingebauten Fehlertoleranzmechanismen die Laufzeiten unzulässig beeinflussen.

In der Diskussion stand der Reifegrad des Ansatzes im Mittelpunkt. Die Teilnehmer interessierten sich hauptsächlich für gesammelte Erfahrungswerte, mögliche Erweiterungen, gegebenenfalls vorliegende Domänenabhängigkeiten des Ansatzes. Auch die Möglichkeit des Einsatzes im Rahmen einer Zertifizierung wurde besprochen. Des Weiteren wurde die Unterscheidung zwischen Hardware- und Softwarefehlern in Bezug auf den Ansatz diskutiert.

4.3 Beitrag 3: A pattern-based requirement specification language: Mapping automotive specific timing requirements

Der dritte Vortrag „A pattern-based requirement specification language: Mapping automotive specific timing requirements“ gehalten durch Philipp Reinkemeier, befasste sich mit dem Vergleich einer musterbasierten Anforderungsbeschreibungssprache (RSL) mit der Sprache Timing Augmented Description Language (TADL) und zeigte, dass die Ausdrucksmöglichkeit der formalen RSL ausreicht, um die durch die TADL unterstützten Zeiteigenschaften abbilden zu können.

In der Diskussion erörterten die Teilnehmer, in welchen Entwicklungsphasen der Einsatz der RSL sinnvoll ist. Einerseits wurde überlegt, wie viel Lösungsaspekte in den Patterns bereits enthalten sind und ob durch den Einsatz die Gefahr besteht, bestimmte Architekturentscheidungen zu beeinflussen.

4.4 Beitrag 4: Herausforderungen für das Requirements Engineering in der Domäne Energie

In dem vierten Vortrag zum Beitrag „Herausforderungen für das Requirements Engineering in der Domäne Energie” stellte Kim Lauenroth spezifische Eigenschaften moderner intelligenter Stromnetze (Smart Grids) dar und zeigte auf, welchen Einfluss die Entwicklung zum „Smart Grid“ auf die Methoden des Requirements Engineerings für eingebetteten Systeme in dieser Domäne haben.

In der Diskussion wurden die beschriebenen Herausforderungen auf vergleichbare Domänen übertragen. Unter anderem wurden Vergleiche mit der Komplexität des Internets und von Kommunikationsnetzen angeführt. Des Weiteren wurde die Notwendigkeit der Interdisziplinarität bei der Entwicklung von Embedded Systems diskutiert, bspw. die enge Zusammenarbeit mit Spezialisten aus den Bereichen Maschinenbau, Physik und Elektrotechnik.

4.5 Beitrag 5: Modellgetriebene Validierung von System-Architekturen

Der fünfte Vortrag „Modellgetriebene Validierung von System-Architekturen", gehalten durch André Pflüger, befasste sich mit einem Ansatz zur Validierung von Architekturen. Das Ziel der vorgestellten Arbeit ist die Reduzierung nicht-kreativer und zeitintensiver Tätigkeiten von System-Architekten im Rahmen der Validierung von Architekturen. Als Besonderheiten des Ansatzes wurden die explizite Modellierung von Validierungszielen und die Auswahl eines geeigneten Validierungsmodells angeführt.

In der Diskussion hinterfragten die Teilnehmer das Ziel einer Kostenreduzierung und erörterten, inwieweit der Zusatzaufwand für die Erstellung der Validierungsziele und Validierungsmodelle, durch Teilautomatisierung der manuellen Tätigkeiten ausgeglichen werden kann. Außerdem wurde die Bedeutung der Spezifikation der Reihenfolge von Abhängigkeiten in der Architekturabbildung diskutiert.

4.6 Offene Diskussionsrunde: Erwartungen an Embedded System im Jahr 2020

Das initiale Thema der Diskussionsrunde wurde mit „Erwartungen an Embedded Systems im Jahr 2020" umschrieben. Im Rahmen der Diskussion wurden von den Teilnehmern insbesondere die folgenden Aspekte thematisiert:

- Die Komplexität von eingebetteten Systemen wird in den kommenden Jahren kontinuierlich zunehmen. Dies begründet sich zum einen aus dem wachsenden Funktionsumfang, der durch eingebettete Systeme realisiert werden soll, und zum anderen durch die zunehmende Vernetzung und Integration von eingebetteten Systemen zur Realisierung innovativer Funktionalitäten. Daher sind Entwicklungsansätze zur Beherrschung dieser Komplexität von großer Bedeutung.

- Kosten- und Zeitdruck wird in den kommenden Jahren weiterhin ein Thema bleiben, insbesondere zur Wahrung der Wettbewerbsfähigkeit in der Industrie. Möglichkeiten und Wege zur Effizienzsteigerung bleiben damit weiterhin eine große Herausforderung.

- Eingebettete Systeme werden typischerweise über einen Zeitraum von vielen Jahren hinweg kontinuierlich weiterentwickelt. Hierdurch entsteht in den Unternehmen ein großer Wert in Form von dokumentiertem Wissen, u.a. in Form von Spezifikationen und insbesondere in Form von Quellcode. Dieses Wissen ist aufgrund des Umfangs teilweise nur schwer zugänglich. Eine zentrale Herausforderung ist daher die Entwicklung von Vorgehensweisen zur Analyse und Auswertung, um dieses Wissen für die Unternehmen leichter verfügbar zu machen.

- In Bezug auf neue Entwicklungsansätze wurde insbesondere der Aspekt der Einführung neuer Ansätze in Unternehmen thematisiert. Die besondere Herausforderung besteht aus Sicht der Industrie darin, die neuen Ansätze derart zu gestalten, dass sie möglichst reibungslos in die bestehenden Prozesslandschaften der Unternehmen eingeführt werden können.

5 Ausblick und Danksagung

Die Beiträge und Diskussionen bei ENVISION 2020 zeigen, dass die Forschung vielversprechende Konzepte anbietet und die Industrie ein großes Interesse an der Erprobung und Einführung dieser Konzepte hat. Obwohl das Gebiet in den letzten Jahren bedeutende Fortschritte erzielt hat, sind für eine durchgängige, modellbasierte Entwicklungsmethodik für softwareintensive eingebettete Systeme noch zahlreiche offene Fragen zu klären und grundlegende Herausforderungen zu überwinden.

Die offene Diskussionsrunde im Rahmen von ENVISION 2020 hat bspw. deutlich gemacht, dass die Berücksichtigung bestehender Produkte und Systeme durch neue Entwicklungsmethoden eine große Rolle für den Erfolg neuer Entwicklungsmethodiken spielt.

Die Resonanz der Teilnehmer zum zweiten ENVISION 2020 Workshop war durchweg positiv. Ein Großteil der Teilnehmer sprach sich für eine Fortführung des Workshops in 2012 aus.

Wir sprechen an dieser Stelle ein herzliches Dankeschön an alle aus, die zum Gelingen dieses Workshops beigetragen habe. Ein besonderer Dank geht an die Organisatoren der SE2011, die ein hervorragendes Umfeld für ENVISION 2020 geboten haben.

A Seamless Model-Based Development Process for Automotive Systems*

Jörg Holtmann, Jan Meyer, Matthias Meyer
s-lab – Software Quality Lab, Software Engineering Group,
Heinz Nixdorf Institute, University of Paderborn
Warburger Str. 100, 33098 Paderborn, Germany
[jholtmann|jmeyer|mmeyer]@s-lab.upb.de

Abstract: Safety critical functions for embedded systems are increasingly realized with software. Current and future standards and maturity models impose high accuracy and quality for the development process of such software-intensive, embedded systems. But nowadays, there are process and tooling gaps between different modeling aspects for the system under development (SUD). Furthermore, the SUD is usually verified and validated not until it is completely implemented, which leads to expensive corrections. In this paper we present a seamless, model-based development process, which is intended for the automotive supplier domain and conforms to the process reference model of Automotive SPICE®[1]. The development process addresses the issues mentioned above by using systematic transitions between different modeling aspects and simulations in early development stages.

1 Introduction

It is no longer possible to imagine our daily lifes without embedded systems. They can be found in different products ranging from home appliances to complex transport systems. The part of functionality realized by software steadily increases in these systems. Consequently, also the number of safety critical functions realized by software grows, especially in the automotive sector.

To achieve high-quality software components realizing such functions, specific procedures are necessary. On the one hand, in addition to functional requirements there are safety requirements that must be fulfilled by a safety-critical component. These safety requirements can be derived from international standards like the IEC 61508 or, for the automotive industry, the upcoming standard ISO 26262. For example, for a component with a high automotive safety integrity level (ASIL), the data exchange via its interfaces has to be secured using CRC-checksums. Such safety requirements are, however, not restricted to the system under development (SUD) but also concern the development process. The fulfillment of the safety standards is mandatory by law.

*This work was partially funded by the German Federal Ministry of Education and Research (BMBF), grant "SPES2020, 01IS08045H".

[1] Automotive SPICE® is a registered trademark of the Verband der Automobilindustrie e.V. (VDA).

On the other hand, the conformity to maturity models is a competitive advantage. In the automotive domain, the original equipment manufacturer (OEM) often requests a certain level of the Automotive SPICE® maturity model for their suppliers. In addition to the requirements imposed by safety standards, the process reference model of Automotive SPICE® [Aut10] demands properties like traceability and defines development stages as well as resulting artifacts. But since standards and process reference models have to be generally applicable to a broad range of organizations, they do not specify how the realization of the properties or artifacts can be achieved.

Models have found their way into current development processes of embedded systems, also in the automotive sector. However, often different specialized models for specific aspects are used at different development stages with varying abstraction levels in a fractal way [TRS+10, Bro06]. This leads to several problems. First of all, there are process and tooling gaps between these models. It is unclear how to move from, for example, textual requirements to model-based design [WW03] and from the system architecture to the software architecture [DN08]. Thus, there is a need for integrated model chains [Bro06]. Second, traceability between different models and between models and requirements is nowadays established by manually linking the individual model elements and requirements. As the amount of requirements and hence the size of the models grow, the lack of automation for this task is increasingly a problem. Third, there is no guidance which models have to be used at a specific development stage [TRS+10]. Process reference models are too general to help on this problem, so a concrete development process is necessary.

In this paper, we address these problems by defining a seamless, model-based development process conforming to Automotive SPICE®, which uses systematic and partially automated transitions between different modeling perspectives.

Since embedded systems consist of hardware and software parts, the interaction between these diverse elements is often not tested until the end of the development process when all software and hardware components are available. If there are errors in the SUD, their correction is very expensive. Thus, a verification and validation (V&V) of the design decisions in early development stages is necessary. We will show, how the system architecture can be extended to include all necessary information for simulations of the SUD already in early stages of our development process.

As an ongoing example for an embedded system we are using a comfort control unit. This electronic control unit (ECU) is responsible for the control of the interior light, the central locking, and the adjustment of windows, mirrors, and seats. The comfort control unit is not a high-grade safety-critical system, but it is in connection with other systems that have a high safety level. Thus, it also contains some functions to check the interior whether persons are aboard. This information is necessary for the airbag system. Since this is a high-grade safety-critical system, the comfort control unit (as it is a data provider) has also a classification as a safety-critical system.

In the next section we present a systematic development process for the automotive supplier domain and show the different artifacts, extensions, techniques, and their usage in the development process. In Section 3 the related work is discussed. At the end of the paper a conclusion and an outlook is presented.

2 Seamless Model-Based Development of Automotive Systems

We base our systematic development process for automotive systems on the Automotive SPICE® process reference model (PRM), which defines ten so-called *engineering processes* (ENG). Figure 1 depicts those processes organized in a V-Model. In the following three subsections we present techniques supporting the engineering processes in the left part of the V-Model. Additionally, these techniques will be associated with modeling perspectives similar to the ones defined in [TRS+10]. These perspectives cover different modeling aspects of the SUD and thus serve as means to achieve separation of concerns.

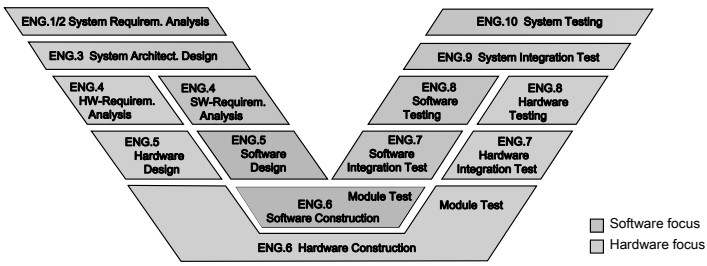


Figure 1: A development process from the automotive supplier domain following the Automotive SPICE® process reference model

The development starts with informal textual requirements on OEM side (ENG.1). Our approach starts with the manual creation of formalized, textual requirements, their validation, and the transition to the model-based system architectural design (cf. Subsection 2.1). Thus, this subsection covers the engineering process ENG.2 and the transition to ENG.3. In ENG.3 the system architecture including a first specification of the operating system (OS) is done. It can be simulated wrt. real-time and resource requirements (cf. Subsection 2.2). Later it is enriched with further details like a first estimation for software resource consumption (ENG.4 and ENG.5). Finally, the transition from the architecture model to the software design specified with the the architecture framework and standard AUTOSAR [AUT09] is ensured by using model transformation and code synthesis techniques (ENG.5 and ENG.6), which is presented in Subsection 2.3. Furthermore, we present a simulation for the AUTOSAR model additional to the simulation in 2.2.

2.1 Formalization of requirements and transition to model-based design

As stated in the introduction, model-based design is applied in current development processes for automotive systems. In contrast, requirements for them are nowadays mainly specified in unrestricted natural language. This is due to the fact that natural language is easy to use, because it does not require training or dedicated tools [Poh10], so all stakeholders can understand requirements formulated in this way. But since unrestricted natural language is informal and ambiguous, such requirements cannot be automatically processed

without resolving the ambiguity by hand. Thus, all further tasks like the validation of the requirements, the transition to the model-based design, and the maintenance of traceability have to be done manually, which is time-consuming, error-prone, and often repetitive.

Figure 2 sketches the requirements engineering within our development process. In the automotive domain mostly informal *customer requirements* (e.g., unrestricted natural text, drawings) as well as a *communication matrix* containing the communication bus signals to the surrounding systems are delivered by the OEM. The customer requirements specify behavior that is observable by the end users (i.e., the car passengers), and thus reside in the user perspective. From the viewpoint of the supplier, this is the Automotive SPICE® engineering process ENG.1 (requirements elicitation, cf. Figure 1). In the process of system requirements analysis (ENG.2), these customer requirements are manually analyzed by the supplier and *system requirements* are created, which propose a possible solution offering the required system functionality. Thus, this process belongs to the functional perspective. The system requirements analysis is one of the most important engineering processes, since the system requirements serve as basis for all further engineering processes [HMDZ08].

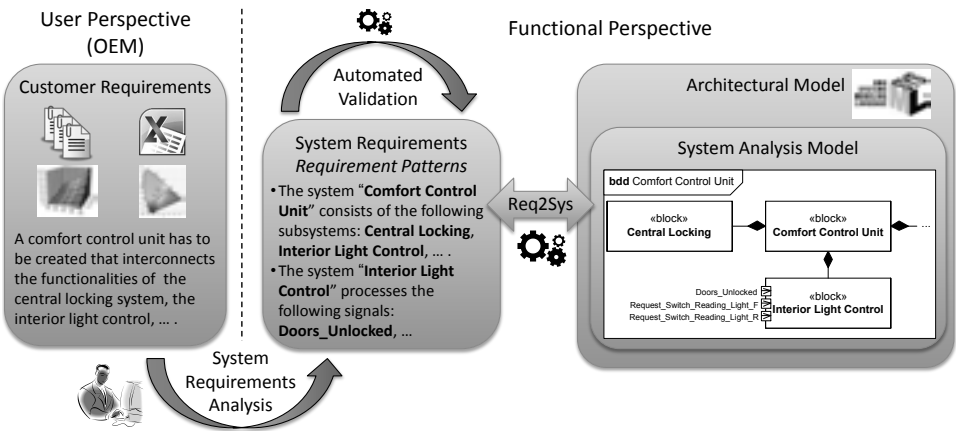


Figure 2: Requirements engineering using requirement patterns

To overcome the problems caused by the use of natural language for the formulation of requirements, we propose a controlled natural language (CNL) approach for the specification of functional system requirements in the automotive domain. The CNL restricts the expressiveness of natural language and disambiguates it, enabling automatic processing of the requirements while having textual requirements understandable for all stakeholders at the same time. We use a slightly extended CNL called *requirement patterns* that is already successfully used in the automotive industry [KK06]. Requirements specified with this CNL describe a subsystem and function hierarchy including required inputs and provided outputs for the subsystems and functions. For example, the system requirements in Figure 2 specify that the Comfort Control Unit contains the subsystem Interior Light Control (among others), which reacts to the signal Doors_Unlocked. Furthermore, a manual mapping of such logical signals to the technical signals of the communication matrix is

created. By this means it is checked whether all required information is delivered by the surrounding systems.

Firstly, the possibility of automatically processing the system requirements formulated with the requirement patterns enables to use an automated requirements validation. The validation detects requirements that violate rules for the function hierarchy (like unused signals, a not-well-formed function hierarchy, overlapping (de-)activation conditions for functions, range violations) and proposes corrections [HMvD11]. Secondly, by using parsing and model transformation techniques, we can transfer the information already contained by the system requirements to an initial system analysis model (cf. Req2Sys in Figure 2). This is the starting point of the model-based design and subject of further manual refinement. For example, the subsystems have been translated to SysML blocks whose hierarchical relationships are shown by a block definition diagram. The model transformations are realized in a research prototype and base on bidirectional Triple Graph Grammars (TGGs) [Sch95], which allow—besides the transformation to SysML—synchronization of system requirements and system analysis model as well as a translation to the text back again, if combined with code synthesis techniques. Additionally, since the transformation relates the requirements with the model elements, we automatically gain traceability. Further details are given in [Hol10].

2.2 Modeling and simulation of the system architecture

The next step in the development process according to Automotive SPICE® is the creation of the *system architecture design*. This corresponds to the engineering process ENG.3 of the Automotive SPICE® PRM (see Figure 1). We use the systems modeling language (SysML) to specify the hardware and software subsystems including their interrelations. Additionally, in this engineering step the decision has to be made which functions will be implemented in hardware and which will be implemented in software [HMDZ08]. As input the system requirements and the system analysis model from the system requirements analysis are used and refined in a manual way to get the system architecture design. Figure 3 depicts our view on the system architecture design as part of the architecture model. In the example, the Interior Light Control subsystem has been refined to contain three parts with the stereotype AtomicSWC (necessary for the model transformation described in the next subsection), as shown by the internal block diagram (ibd Interior Light Control).

Besides the structural view of the system, its behavior has to be specified, too. For embedded systems, the behavior is often characterized by timing requirements. Since SysML is lacking appropriate means to specify timed behavior, we extended SysML by *Real-Time Statecharts (RTSCs)* [HMSN10]. RTSCs extend the UML/SysML State Machines on the basis of Timed Automata, which facilitates to model reactive and timed behavior in a formal way. Additionally, RTSCs can be formally verified on fulfillment of timing requirements. One example for a timing requirement of the comfort control ECU is: “The interior light has to be on for 3000 ms after the doors are unlocked.” The RTSC representing this informal requirement is shown in Figure 3. It contains two states representing Light on and Light off. Upon receiving a Doors_Unlocked event, the light is switched on or

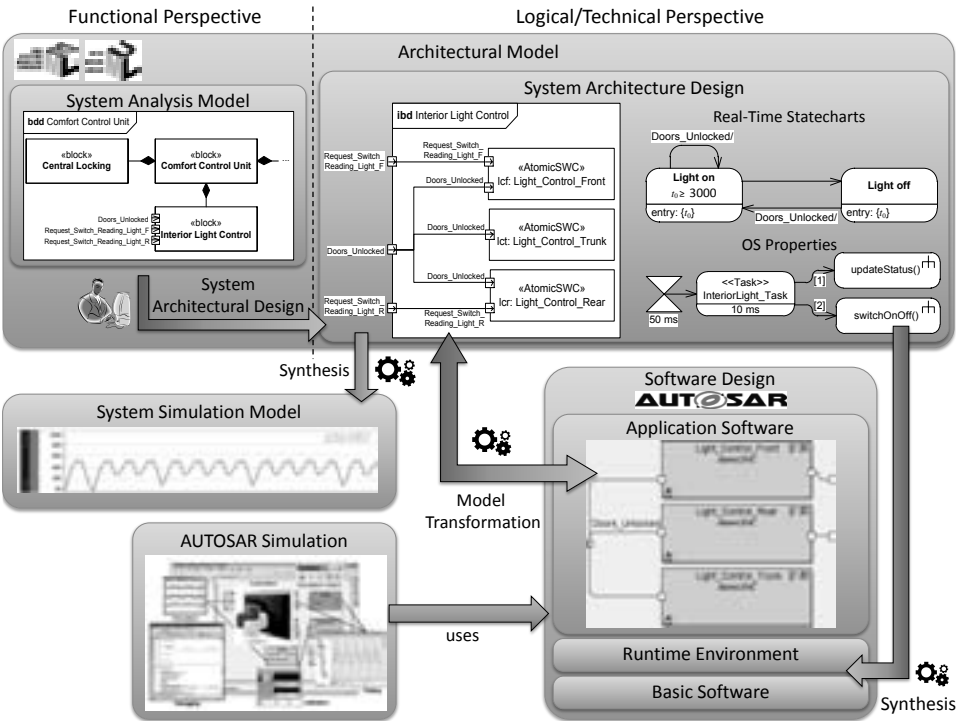


Figure 3: The system architecture design and its extensions, the transition to the software design, and the usage of simulations

stays on. The clock variable t_0 is used to express the timing: Light on may only be left after 3000 ms (specified by the time invariant $t_0 \geq 3000$) and t_0 is reset to 0 upon entering one of the states (specified by $entry : \{t_0\}$ in both states).

Since we want to allow an early verification of the system and the architectural decisions taken so far, we include a system simulation already in this engineering process (ENG.3), which allows to check, for example, whether the bus or CPU load is acceptable and whether timing requirements are satisfied. The architectural model already contains most of the information required for such a simulation. However, it requires additional information concerning the configuration of the OS running on the ECUs.

In the automotive domain, the OS have no dynamic parts (e.g., OSEK OS). This means that all OS settings are static and can be specified during the development phase. This enables the modeling of OS properties in the architecture model. However, since the SysML has no elements for the specification of OS properties, we extended it with a profile to specify, for example, tasks with their priority and duration, the mapping of functions to tasks, and their activation policies. The profile also supports the specification of resources, alarms and interrupts. As an example, Figure 3 depicts a specification of the cyclic task InteriorLight_Task, which is triggered every 50 ms. Each time the task may run at most 10 ms. During each run, it first executes the function updateStatus(), which checks the door

status and generates events upon a status change. Afterwards, the function switchOnOff() is called, which executes the RTSC depicted above.

For the simulation, we use the real-time simulation tool chronSIM[2]. The tool needs a dedicated simulation model, which contains information about the hardware architecture, the software components, the deployment of software to hardware, and the OS settings; that is, the information we specify in the extended architecture model. Thus, we use a model to text transformation to generate the necessary System Simulation Model for chronSIM as indicated in Figure 3. A detailed description of the transformation and how the simulation output looks like can be found in [NMK10].

The system architecture design covers different aspects of information and changes during the development process. In an early development stage (i.e., the beginning of ENG.3), the system architecture design is defined by a network of interacting logical components and thus belongs to the logical perspective. In later development stages (i.e., the end of ENG.3 to ENG.5), the system architecture design is enriched with technical information like hardware, deployment, and OS details and thus is covered by the technical perspective.

2.3 Transition from the system architecture to AUTOSAR

A further gap in the development process exists between the system architectural design (ENG.3) and the software design (ENG.5). Automotive SPICE® demands another process between ENG.3 and ENG.5, namely the software requirements analysis (ENG.4). In ENG.4 the system requirements that are relevant for software elements are assigned to the individual software items and afterwards analyzed and refined, if required [HMDZ08]. Therefore, the system analysis model (cf. Figures 2 and 3) is refined with, for example, Activity, Sequence or (Real-Time) Statechart Diagrams. These requirements are used as an input for the software design (ENG.5), which we describe with AUTOSAR [AUT09].

An AUTOSAR architecture consists of three major layers. The first one contains the application software; that is, the application software components and their algorithms, itself. The second layer is the so called Runtime Environment (RTE), which serves as a middleware for the communication of software components potentially located on different ECUs. The RTE is automatically created by code generators. The third layer is the so-called basic software, which contains partially hardware-dependent software providing basic services to the application software. Examples for the basic software are the OS or the communication stack, which is responsible for the communication to other ECUs.

In order to establish a seamless development from the requirements via the architectural model to the software design, a further model transformation from the architectural model to AUTOSAR is necessary. Our transformation consists of two parts. In the first part, the structural elements like components with ports and connectors are transformed to AUTOSAR [GHN10]. In Figure 3 this is shown with the arrow Model Transformation: the three parts lcf:Light_Control_Front, lct:Light_Control_Trunk, and lcr:Light_Control_Rear are transformed into atomic software components in AUTOSAR. Like the model transforma-

[2] http://www.inchron.com/chronsim.html

tions between system requirements and system analysis model, the model transformations between SysML and AUTOSAR application components base on TGGs. Their bidirectional nature enables synchronization of both models and round-trip capabilities. This is one of the major advantages over approaches using SysML and AUTOSAR in a complementary but manual way: if changes on one model occur, they can be transferred to the other model automatically in order to reestablish consistency.

The second part is responsible for the configuration of the RTE and the basic software. The functions like updateStatus() or switchOnOff() specified in the architectural model, can be translated to so-called *runnables* in AUTOSAR. Those runnables are used in turn for the RTE generation. This can also be done for further elements of the RTE like exclusive areas, interrunnable variables, and so on. The configuration of the basic software can be described by transforming the OS properties specified in the architectural model. In the transformation step, the tasks from the architectural model are transformed to AUTOSAR OS tasks. In the same way, more parts of the AUTOSAR basic software can be configured, for example, the communication stack or parts of the memory stack. The transformation closes the gap to the software design (ENG.5). The transition to software construction (ENG.6) is done with the help of AUTOSAR code generators and by using the RTE API.

However, besides the automated model transformation, some manual refinements of the AUTOSAR software architecture are still necessary, for example, the decision whether implicit or explicit communication is used. Furthermore, the generated RTE has a big influence on the AUTOSAR software architecture. This is an additional layer that has to be verified and validated in interaction with the application components and the basic software. Consequently, another simulation is necessary to verify the AUTOSAR architecture. For this purpose, we use the AUTOSAR authoring and simulation tool SystemDesk[3]. This simulation shows, whether the RTE is configured correctly and timing requirements are still fulfilled. For example, it is verified whether the RTE can handle the application data as fast as it is necessary to switch on the lights after a door is unlocked.

3 Related Work

There are some other approaches that deal with improving the development process in the automotive domain and close the process gaps. In [Kor07] and [Bol09] a systematic modeling with SysML and AUTOSAR is presented. The approaches suggest to use the SysML and AUTOSAR in a systematic development process, like we do in our approach. The gap between the two modeling languages is not closed by means of automatic model transformations, but by manually establishing traceability. Therefore, the developer has to manually model both architectures and afterwards create traceability links. This fulfills the requirements of Automotive SPICE®, but it does not speed up the development process.

The tool platform EDONA [OT10] aims at integrating the architecture description language EAST-ADL2 [ATE10] (also based on SysML and UML) and AUTOSAR by using

[3] http://www.dspace.com/en/pub/home/products/sw/system_architecture_software/systemdesk.cfm

model transformations. A further aim related to our work is a simulation of generated code including the OS properties. In contrast to our approach, EDONA focuses only on techniques and tools but does not consider any process reference model. Furthermore, the transition from textual requirements to the model-based design and the traceability between them is not addressed. Finally, the transition from the system model to AUTOSAR encompasses only the component descriptions on the application layer. Thus, the generation of parts of the basic software configuration is not supported.

4 Conclusion and Outlook

In this paper we presented a seamless and systematic model-based development process for automotive systems. We explained how system requirements written in natural language can be formalized. This way, we enable an automated processing of the requirements; that is, a validation and the extraction of information into an initial system analysis model, while all stakeholders are still able to understand them. Furthermore, we extended the modeling languages with notations to specify timed behavior and OS properties. On the one hand, this facilitates the generation of simulation models for V&V of the system in early design stages. On the other hand, the extended architectural model enables the transformation of architectural elements to an AUTOSAR model.

As explained in the introduction, Automotive SPICE® does not provide any guidelines which modeling language shall be used in an engineering process for the creation of an artifact. Thus, our development process can be seen as a possible instance of the Automotive SPICE® PRM, which additionally provides guidance for the developers on moving from the system requirements via the system architecture design and the software design to the implementation. Furthermore, the presented simulation techniques allow the V&V of the SUD at early design stages instead of implementing and afterwards verifying and validating a complete prototype consisting of software and hardware. Thus, since errors are detected earlier, this leads to less expensive corrections.

One of our further research goals is a tighter connection between the requirement patterns and the system real-time simulation, so that the requirements can be used as direct input for the simulation. Secondly, the requirement patterns describe a function hierarchy, but currently we do not specify how a function shall work. When we extended the CNL in such a way, it would be possible to check whether there is an already implemented component that realizes the function—and hence boosting reuse of components. As we mentioned the upcoming standard ISO 26262, another goal is to enhance our development process to easily integrate techniques for ensuring functional safety. Finally, we want to extend our current demonstrator models for a more extensive evaluation of the overall process.

References

[ATE10] ATESST2. EAST-ADL Domain Model Specification. Version 2.1 RC3, 2010.

[AUT09] AUTOSAR GbR. AUTOSAR Specification, 2009.

[Aut10] Automotive Special Interest Group (SIG). Automotive SPICE®: Process Reference Model. Release v4.5, 2010.

[Bol09] Richard F. Bolt. Modeling AUTOSAR systems with a UML/SysML profile. Technical report, IBM Software Group, Rational, 2009.

[Bro06] Manfred Broy. Challenges in Automotive Software Engineering. *International Conference on Software Engineering (ICSE)*, 2006.

[DN08] Marco Di Natale. Design and Development of Component-Based Embedded Systems for Automotive Applications. In *Reliable Software Technologies – Ada-Europe 2008*, volume 5026 of *LNCS*, pages 15–29. Springer, 2008.

[GHN10] Holger Giese, Stephan Hildebrandt, and Stefan Neumann. Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. In *Graph Transformations and Model-Driven Engineering*, volume 5765 of *LNCS*, pages 555–579. Springer, 2010.

[HMDZ08] Klaus Hörmann, Markus Müller, Lars Dittmann, and Jörg Zimmer. *Automotive SPICE in Practice: Surviving Implementation and Assessment*. Rocky Nook, 2008.

[HMSN10] Jörg Holtmann, Jan Meyer, Wilhelm Schäfer, and Ulrich Nickel. Eine erweiterte Systemmodellierung zur Entwicklung von softwareintensiven Anwendungen in der Automobilindustrie. In *Software Engineering 2010 – Workshopband*, volume 160 of *LNI*, pages 149–158. Bonner Köllen Verlag, 2010.

[HMvD11] Jörg Holtmann, Jan Meyer, and Markus von Detten. Automatic Validation and Correction of Formalized, Textual Requirements. In *Proceedings of the ICST Workshop Requirements and Validation, Verification & Testing (ReVVerT 2011)*, 2011. Accepted.

[Hol10] Jörg Holtmann. Mit Satzmustern von textuellen Anforderungen zu Modellen. *OBJEKTspektrum*, RE/2010 (Online Themenspecial Requirements Engineering), 2010.

[KK06] Roland Kapeller and Stefan Krause. So natürlich wie Sprechen - Embedded Systeme modellieren. *Design & Elektronik*, 08:64–67, 2006.

[Kor07] Andreas Korff. AUTOSAR und UML – das passt. *Automobil Elektronik*, 2007.

[NMK10] Ulrich Nickel, Jan Meyer, and Tapio Kramer. Wie hoch ist die Performance? *Automobil-Elektronik*, 03:36 – 38, 2010.

[OT10] François Ougier and François Terrier. Eclipse based architecture of the EDONA platform for automotive system development. In *Proceedings of the European Congress on Embedded Real-Time Software and Systems (ERTS^2 2010)*, 2010.

[Poh10] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, 2010.

[Sch95] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In *Graph-Theoretic Concepts in Computer Science*, volume 903 of *LNCS*, pages 151–163. Springer, 1995.

[TRS+10] Judith Thyssen, Daniel Ratiu, Wolfgang Schwitzer, Alexander Harhurin, Martin Feilkas, and Eike Thaden. A System for Seamless Abstraction Layers for Model-based Development of Embedded Software. In *Software Engineering 2010 – Workshopband*, volume 160 of *LNI*, pages 137–147. Bonner Köllen Verlag, 2010.

[WW03] Matthias Weber and Joachim Weisbrod. Requirements Engineering in Automotive Development: Experiences and Challenges. *IEEE Software*, 20:16–24, 2003.

Timing Overhead Analysis for Fault Tolerance Mechanisms

Kai Höfig

hoefig@cs.uni-kl.de

Abstract: The growing complexity of safety-critical embedded systems is leading to an increased complexity of safety analysis models. Often used fault tolerance mechanisms have complex failure behavior and produce overhead compared to systems without such mechanisms. The question arises whether the overhead for fault tolerance is acceptable for the increased safety of a system. In this paper, an approach is presented that uses safety analysis models of fault tolerance mechanisms and execution times of its subcomponents to generate *failure dependent* execution times. This provides a detailed view on the safety behavior in combination with the produced overhead and allows a trade-off analysis to find appropriate fault tolerance mechanisms.

1 Introduction

Safety-critical embedded systems are ubiquitous in our daily lives and can be found in many sectors, such as automotive, aerospace, medicine, automation, and energy industries. They are becoming more and more complex due to increasing functionality and automation in industry. The corresponding safety analysis models also grow in complexity and level of detail. To increase the safety of such systems, redundancies are often used within a certain mechanism to tolerate faults in the redundant units. These so-called *fault tolerance mechanisms* are widely used concepts and have a known behavior. They produce an overhead, e.g., in terms of execution time, energy consumption, additional hardware, additional software, or additionally required components compared to systems without such a mechanism. The problem arises whether the overhead produced by a fault tolerance mechanism is acceptable for the increased safety of a system.

Since safety requirements often contain a deadline and an upper bound for failure probability, e.g., *the system has to provide its function within x ms with a failure probability of less than y*, this paper addresses the property of *overhead in time* of fault tolerance mechanisms. The central approach presented in this paper aims at the fact that a fault tolerance mechanism is in a certain *mode* with a specific execution time for some faults being tolerated. The mode itself depends on failure modes given by safety analysis models. The combination of the time consumed by a mode and failure modes provides a detailed prospect of the timing behavior for a fault tolerance mechanism. Thereby a trade-off analysis in terms of execution time is supported.

In section 2, related approaches are described. Since failure modes of safety analysis models are here combined with execution times, that section is divided into approaches which belong to the research area of *Worst Case Execution Time* and into approaches that sup-

port the process of *safety engineering*. In section 3, an example system is described that is used to introduce the safety analysis model of *Component Fault Trees*, which provides failure modes as an input for the approach presented in this paper. Section 4 is the central section of this paper. The example system is picked up to describe the problem of modeling failure-dependent overhead in time manually. The central approach of generating execution times for fault tolerance mechanisms according to failure modes of the safety analysis model is formalized and applied to the example system. The generated execution times allow a sophisticated view of the overhead in time for such mechanisms. Section 5 concludes this paper and provides a perspective for future work.

2 Related Work

In this section, related approaches are discussed. Since the approach presented in this paper crosses two research areas, this section is divided into approaches that belong to the field of *Worst Case Execution Time* and into approaches that belong to the research area of *Safety Engineering*.

Current approaches to WCET analysis can be divided into measurement-based approaches and static timing analysis approaches [WEE+08]. In static timing analysis, the execution times of individual static blocks are *computed* for a given program or a part of it. In general, these approaches provide safe upper bounds for the WCET by making pessimistic assumptions at the expense of overestimating the WCET in order to guarantee deadlines for the analyzed program. Advanced approaches, e.g. like those presented in [FH04], encompass precise hardware models to reduce overestimation of the WCET as much as possible. On the other hand, measurement-based approaches do not need to perform any complex analysis. They *measure* the execution time of a program on real hardware or processor simulators. These approaches can, in general, not provide a safe upper bound for the WCET, since neither an initial state nor a given input sequence can be proven to be the one that produces *the* WCET. Both static timing analysis and measurement-based approaches do not encompass additional timing failure modes or failure probabilities for calculating probabilistic WCETs for several modes of an analyzed system. However, approaches can be found that split the WCET in a probabilistic way. In [BCP03], an approach is presented that calculates probabilistically distributed WCETs. Here, the authors concentrate on *how* the nodes of a syntax tree have to be calculated if the WCETs for its leaves are *given* with probabilistic distribution. In a later work, the approach is extended to also encompass dependence structures. The authors solved the arising problem of multivariate distributions by using a stochastic tool for calculating a probabilistically distributed WCET. To *determine* the probability distribution, there have been some initial approaches in probabilistic WCET analysis. In [BE00], the authors use a measurement-based approach. The central idea is to measure the timings of a task and to stochastically drive the statement that a WCET will provide an upper bound for a certain subset of the input space. This approach is extended for scheduling in [BE01]. In [BBRN02] and [NHN03], probability distributions are calculated for response times of the *Controller Area Network* (CAN) Bus. These approaches may provide input to the approach presented in [BCP03], but do not aim at

calculating WCETs. In contrast, the approach presented in this paper *generates* timing failure modes for fault tolerance mechanisms. This approach can therefore be taken into account as input for previously described approaches such as [BCP03].

Current approaches that automatically deduce safety analysis models from an annotated system development model mainly shift the complexity from the safety analysis model to the system development model. These approaches solve the problem of inconsistencies between the two models by combining the failure logic with the system development model entities by using annotations [PM01, Gru06, Rug05]. Other approaches, like [Boz03] and [JHSW06], rely on a formally executable design model to find inconsistencies between the model and its specification. This procedure allows a high degree of automation, but the type of system model is quite limited and the approaches do not solve the problem of finding appropriate failure modes. Only a few approaches deduce failure behavior by semantically enriching the system development model. In [GD02], an approach is presented that supports the analysis at a high level by providing recurring safety analysis model constructs mainly for redundancy mechanisms. These constructs decrease the complexity of a manual safety analysis, but do not provide a solution for generating timing failure modes. In [dMBSA08], a larger design space is covered, but the high degree of detail in the safety analysis model is achieved at the expense of a large number of annotations. Besides that, this approach does not aim at generating failure modes, but is more dedicated to a preliminary safety assessment aimed at designing an architecture that fulfills the safety requirements. The approach presented in this paper belongs to the group of semantic enrichment, since parts of safety analysis models are *used* to deduce timing failures.

In the next section, an example system is introduced along with its safety analysis model as the running example of this paper.

3 Example System

The example system of this paper is a simple fault tolerance mechanism. This section is used to introduce the methodology of *Component Fault Trees*. This safety analysis model relates parts of a fault tree and failure modes to components and ports (see [KLM03]), what makes it an interesting model for combining execution times of components with their failure modes. The approach presented in this paper uses CFTs, but also different safety analysis models, such as *Generalized Stochastic Petri Nets* or *Markov Chains* may provide input for it.

The example system is structured as follows: figure 1 shows on the left side the SysML *Internal Block Diagram* of a sorting mechanism FTSORT that performs a fault tolerant sorting of an array using two different sorting algorithms executed in the components PRIMARY and ALTERNATE. The system FTSORT starts its operation by storing a received array within a checkpoint. Then the checkpointing sends the array to the PRIMARY and triggers this component to start processing. The PRIMARY sends its result to the DECIDER, which checks whether the array is sorted. If the array is sorted, FTSORT has a result. Otherwise, the array is unsorted and has a VALUE failure. The DECIDER triggers the

CHECKPOINTING to restore the input. The checkpointing then sends the restored former input to the ALTERNATE and triggers this component to start processing on the restored input data. The result of the ALTERNATE is then taken as the result of FTSORT.

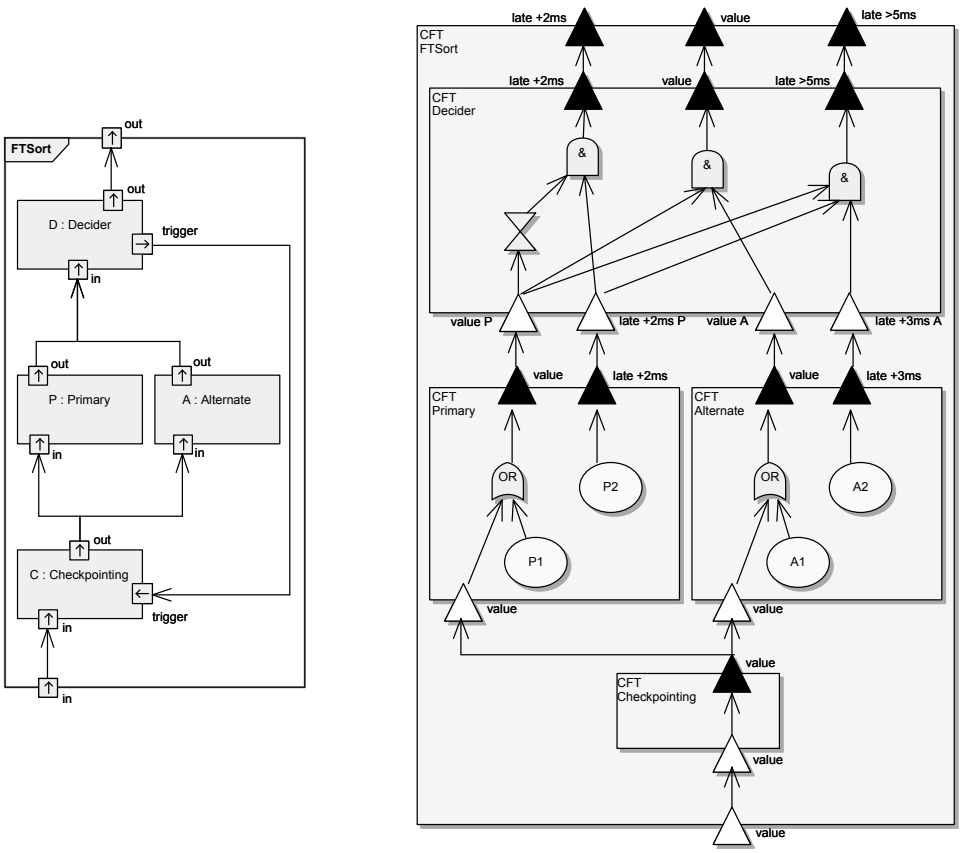


Figure 1: SysML Internal Block Diagram of FTSORT (left side) and related Component Fault Trees (right side).

The safety analysis model for this system is depicted on the right side of figure 1. Component Fault Trees (CFTs) are used to model the failure behavior of FTSORT[1]. Every subcomponent of FTSORT has an associated (sub)CFT. Every CFT has input and output failure modes (triangles). They are associated to the ports of the corresponding component (for example the output failure mode VALUE of the CFT PRIMARY is associated to the OUT port of the component PRIMARY). The port associations are straightforward and therefore not depicted to keep the figure readable. Within a CFT, conventional fault tree gates can be used to model the failure behavior of a component. For example, the com-

[1]For reasons of simplicity, no additional basic failure modes for DECIDER and CHECKPOINTING, a failure propagation via the trigger ports or other failure modes than VALUE and TIMING failures are included in this model.

ponent PRIMARY produces a VALUE failure if the basic failure mode P1 occurs or if it receives data with a VALUE failure (see CFT PRIMARY in figure 1). The system FTSORT produces a VALUE failure, if the basic failure modes P1 and A1 occur or if the provided input has a VALUE failure (see middle AND-gate in the CFT DECIDER in figure 1).

Additional to the VALUE failures, PRIMARY and ALTERNATE have two timing failure modes (see corresponding CFTs in figure 1). If the basic failure mode P2 occurs, the result of the PRIMARY is two milliseconds delayed. If the basic failure mode A2 occurs, the result of the ALTERNATE is three milliseconds delayed. The CFT DECIDER can be used to show that these two timing failure modes are insufficient to provide a sophisticated view on the timing failure modes of the entire mechanism. Both output failure modes LATE+2MS and LATE+3MS are used in this CFT. The DECIDER is delayed by 2 ms if the basic failure mode P2 is active and the PRIMARY component does not produce a VALUE failure (in this cause, the DECIDER would detect a failure and execute the ALTERNATE). This behavior is reflected by the leftmost AND-gate in the CFT DECIDER. Furthermore, the system is delayed by *at least* 5 ms, if the PRIMARY produces a VALUE failure, is additionally delayed by 2 ms and the ALTERNATE is delayed by 5ms (rightmost AND-gate of the CFT DECIDER). Nevertheless, these failure modes cannot be used to express the *overhead in time* for this mechanism, since the significance of the failure mode LATE>5MS, for example, is depending highly on the execution times of PRIMARY, DECIDER and CHECKPOINTING. If, e.g., the PRIMARY requires 500 ms to execute, the actual overhead in time is quite larger then 5 ms.

The safety analysis model indeed provides failure modes that involve a overhead in time, but generating absolute values can become an error prone and time-consuming task. Execution times have to be included in this calculation and additional timing failures complicate this process additionally. This is demonstrated in the next section and the central approach of this paper is presented that automates this process.

4 Analysis of Timing Overhead for Fault Tolerance Mechanisms

To tackle the problem of gaining a sophisticated view on the overhead in time for fault tolerance mechanisms, an approach is described in this section that automatically combines the execution times of components and failure modes of CFTs. As stated in the introduction, there are failure modes in safety analysis models that correspond with a certain *mode* of a fault tolerance mechanism. For example the failure mode VALUE of the component PRIMARY is corresponding to the mode in which the ALTERNATE is invoked to recover from this fault. If this failure mode is not active, the fault tolerance mechanism is in a different mode where only the PRIMARY redundant unit is executed. Such a set is here called a *run*. Since the here presented approach aims at execution times, the sets of executed elements for those two modes are needed as an input. Two run sets for FTSORT are depicted in table 1. The failure mode PRIMARY.OUT.VALUE relates to the value output failure mode of the CFT PRIMARY as depicted in figure 1. The consumed time for a run is based on the execution times of the subcomponents of FTSORT as follows: CHECKPOINTING: 1ms, PRIMARY: 2ms, ALTERNATE: 3ms, and DECIDER: 1ms. The execution

Run	Executed Elements	Consumed Time	Corresponding Failure Mode
1	C,P,D	4ms	**not (**Primary.out.value**)**
2	C,P,D,C,A,D	9ms	Primary.out.value

Table 1: Assumed execution times for the subcomponents of FTSORT

Run	Additional Time	Consumed Time	Corresponding Failure Mode
1	0ms	4ms	**not (**Primary.out.value**)**
1	2ms	6ms	**not (**Primary.out.value**) and** p2
2	0ms	9ms	Primary.out.value
2	2ms	11ms	Primary.out.value **and** p2
2	3ms	12ms	Primary.out.value **and** a2
2	2ms + 3ms	14ms	Primary.out.value **and** p2 **and** a2

Table 2: Possible Timings for FTSORT

times are also an input for this approach. As stated before, the output failure mode VALUE of the component PRIMARY corresponds to both runs.

Since the components PRIMARY and ALTERNATE provide additional timing failure modes LATE+2MS and LATE+3MS, six combinations of different timings can be possible. Those are depicted in table 2. The corresponding failure modes are a conjunction of the failure modes that correspond to a specific run as depicted in table 1 and the failure modes that correspond to an additional timing failure mode. E.g., the first run requires an additional execution time of 2 ms (what sums the consumed time up to 6 ms) if this run is executed *and* P2 is active (second row of table 2). This combination of additional timing failure modes and execution times provides a sophisticated view on the overhead in time produced by the fault tolerance mechanism. For quantified basic failure modes in the safety analysis model and a specific quantified requirement as described in section 1, the table can be used to directly perform a trade-off analysis of the fault tolerance mechanisms in terms of increased system safety and overhead in time.

Nevertheless, for every additional timing failure mode in one of the CFTs of PRIMARY or ALTERNATE, additional alternates, or additional timing failure modes elsewhere in FTSORT, this table will expand rapidly. Calculating a larger number of such combinations manually is error prone and time consuming. Therefore, it is described in the following how this table can be generated if the set of runs is given, the corresponding failure modes are known, and additional timing failure modes are quantified in terms of absolute additional execution time.

Let $\mathcal{V}$ be the set of components that belong to a fault tolerance mechanism with

$$\mathcal{V} = \{V_1, .., V_{\bar{n}}\} \text{ with } |\mathcal{V}| = \bar{n} \in \mathbb{N}.$$

Each component V_i has one associated execution time, here labeled as t_i^0. With all optional additional timing failure modes of V_i, labeled as t_i^k with $k > 0$, the set $\mathcal{T}_i$ represents the

different execution times for the component V_i with

$$\mathcal{T}_i = (t_i^0, .., t_i^{\bar{m}_i}) \text{ with } |\mathcal{T}_i| = \bar{m}_i + 1 \in \mathbb{N}.$$

Each t_i^k is a tuple that consists of a time and a failure mode fm with

$$t_i^k = (time, fm),$$

whereat $fm(t_i^k)$ represents the corresponding timing failure mode of the safety analysis model and $time(t_i^k)$ represents the consumed time if this failure mode is active. If necessary, $time(t_i^k)$ has to be combined with t_i^0 to reflect the execution time of the component in combination with additional required time for the failure mode. Additionally, $fm(t_i^0)$ is set to `true` to ease the later construction.

In the example, the component PRIMARY (P) has two execution times in $\mathcal{T}_P$: t_P^0 and t_P^1 with $time(t_P^0) = 2ms$, $time(t_P^1) = 4ms$, $fm(t_P^0) = true$ and $fm(t_P^1) = late + 2ms$.

These sets of execution times and their corresponding failure modes are used in the following to generate the different execution times of a run. The set $\mathcal{R} = (run_1, .., run_s)$ is the set of runs as described in the example. Each run is a tuple of executed components that belong to this run and corresponding failure modes that determine this run with

$$run_i = ((V_i, .., V_j), failuremodes), V_i, .., V_j \in \mathcal{V}$$

In the previous example, the corresponding sets for the runs as depicted in table 1 have the following form:

$$run_1 = ((C, P, D), not(Primary.out.value))$$

$$run_2 = ((C, P, D, C, A, D), Primary.out.value)$$

Using this sets, the all possible combinations of executions can be deducted. Let the set $\Omega(run_k)$ hold all combinations of execution times of components that are possible for a certain $run_K \in \mathcal{R}$ with

$$\begin{aligned} \Omega(run_k) = \{\omega = (t_m, .., t_n) \quad | \quad & m \leq i \leq n, \\ & run_k = ((V_m, .., V_n), fm), V_i \in \mathcal{V}, \\ & t_i \in \mathcal{T}_i\}. \end{aligned}$$

For the example system, the corresponding set for run_1 is $\Omega(run_1) = ((t_C^0, t_P^0, t_D^0), (t_C^0, t_P^1, t_D^0))$, since only the CFT for the PRIMARY has an additional timing failure mode. To generate tuples of execution times and corresponding failure modes for an ω_i, the execution times are summed up and the failure modes are combined.

In the following, execution times are combined by simply summing them up. It is here important to mention, that this is in general not applicable to approaches that generate *Worst*

Case Execution Times (WCETs), since for two components A and B, $WCET(A+B) \neq WCET(A) + WCET(B)$. This is true for many approaches that encompass processor states and stack values for calculating WCETs. To use such complexer WCET approaches in combination with our approach, the execution time for an entire run has to be calculated by a WCET tool. But to demonstrate our approach, we assume the independence of the given timings and for an $\omega \in \Omega(run_k)$ with $\omega = (t_m, .., t_n), t_i \in \mathcal{T}_i$, the consumed time is

$$time(\omega) = time(t_m) + .. + time(t_n).$$

The corresponding failure mode that models the execution of a specific combination of timing values for a run $\omega \in \Omega(run_k)$, is a conjunction of the specific failure mode of the run, $fm(run_k)$ and all failure modes that are active to result in the specific time behavior. The symbol $\wedge$ is here used to express the Boolean AND. The corresponding failuremode is then

$$fm(\omega) = fm(run_k) \wedge fm(t_m) \wedge .. \wedge fm(t_n).$$

In the example system, we previously constructed the set $\Omega(run_1)$ with $\omega_2 = (t^0_C, t^1_P, t^0_D)$. Since $fm(t^0_C) = fm(t^0_D) = true$, $fm(t^1_P) = p2$ and $fm(run_1) = not(Primary.out.value)$, the corresponding failuremode for ω_2 is:

$$\begin{aligned} fm(\omega_2) &= not(Primary.out.value) \wedge true \wedge p2 \wedge true \\ &= not(Primary.out.value) \wedge p2. \end{aligned}$$

Using this construction, the different execution times are combined with failure modes in a fashion as described for the example system at the beginning of this section. The construction of different alternate executions provides a sophisticated view on the overhead in time of fault tolerance mechanisms. The automated construction of possible alternates is less error prone than a manual approach and is capable to handle a larger amount of combinations and failure modes. In the next section, we conclude this paper and provide an outlook for future work.

5 Conclusion and Future Work

In this paper, an approach is presented that uses timing failure modes of safety analysis models and execution times of *modes* of fault tolerance mechanisms to automatically deduce different execution times along with their corresponding failure modes within the safety analysis model. This sophisticated view on execution times supports a trade-off analysis in terms of safety and overhead in time for fault tolerance mechanisms. Such a trade-off analysis can save costs when fault tolerance mechanisms cannot be identified as an appropriate solution by classic execution time analyses.

Section 3 provides an example for such a mechanism and motivates for combining execution times with safety analysis models to perform a trade-off analysis for the overhead in

time of fault tolerance mechanisms. Since the number of possible combinations may be problematic for a manual approach, we present in section 4 an approach that automatically deduces execution times and corresponding failure modes. This requires clearly identified *modes* of fault tolerance mechanisms.

In our future work we concentrate on extending the approach to generic mechanisms, respectively to parallel redundancy. Furthermore, we want to be able to automatically deduce the execution behavior from a system model. This reduces the effort for applying the here presented methodology and also allows the use of WCET tools that are able to calculate tight upper bounds.

References

[BBRN02] I. Broster, A. Burns und G. Rodriguez-Navas. Probabilistic analysis of CAN with faults. *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, Seiten 269 – 278, 2002.

[BCP03] Guillem Bernat, Antoine Colin und Stefan Petters. pWCET: A tool for probabilistic worst-case execution time analysis of real-time systems. Bericht, 2003.

[BE00] A. Burns und S. Edgar. Predicting computation time for advanced processor architectures. *Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on*, Seiten 89 –96, 2000.

[BE01] A. Burns und S. Edgar. Statistical analysis of WCET for scheduling. *Real-Time Systems Symposium, 2001. Proceedings. 22nd IEEE*, Seiten 215 – 224, dec. 2001.

[Boz03] M. Bozzano. ESACS: An integrated methodology for design and safety analysis of complex systems. In *Proc. of European Safety and Reliability Conf. ESREL*, Seiten 237–245, 2003.

[dMBSA08] M. A. de Miguel, J. F. Briones, J. P. Silva und A. Alonso. Integration of safety analysis in model-driven software development. *Software, IET*, 2(3):260–280, June 2008.

[FH04] C. Ferdinand und R. Heckmann. aiT: Worst-Case Execution Time Prediction by Static Program Analysis. In *Building the Information Society*, Jgg. 156/2004, Seiten 377–383, 2004.

[GD02] P. Ganesh und J.B. Dugan. Automatic Synthesis of Dynamic Fault Trees from UML SystemModels. *13th International Symposium on Software Reliability Engineering (ISSRE)*, 2002.

[Gru06] L. Grunske. Towards an Integration of Standard Component-Based Safety Evaluation Techniques with SaveCCM. *Proc. Conf.Quality of Software Architectures QoSA*, 4214, 2006.

[IEC98] IEC61508. International Standard IEC 61508, 1998. International Electrotechnical Commission (IEC).

[JHSW06] A. Joshi, M.P.E. Heimdahl, M.P. Steven und M.W. Whalen. Model-Based Safety Analysis, 2006. NASA.

[KK07] Israel Koren und C. Mani Krishna. *Fault-Tolerant Systems*. Morgan Kaufmann, San Francisco, 2007.

[KLM03] Bernhard Kaiser, Peter Liggesmeyer und Oliver Mäckel. A new component concept for fault trees. In *SCS '03: Proceedings of the 8th Australian workshop on Safety critical systems and software*, Seiten 37–46, Darlinghurst, Australia, 2003. Australian Computer Society, Inc.

[LPW09] Philipp Lucas, Oleg Parshin und Reinhard Wilhelm. Operating Mode Specific WCET Analysis. In Charlotte Seidner, Hrsg., *Proceedings of the 3rd Junior Researcher Workshop on Real-Time Computing (JRWRTC)*, Seiten 15–18, October 2009.

[NHN03] T. Nolte, H. Hansson und C. Norstrom. Probabilistic worst-case response-time analysis for the controller area network. *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, Seiten 200 – 207, may. 2003.

[PM01] Y. Papadopoulos und M. Maruhn. Model-Based Automated Synthesis of Fault Trees from Matlab.Simulink Models. *International Conference on Dependable Systems and Networks*, 2001.

[Rug05] Ana Rugina. System Dependability Evaluation using AADL (Architecture Analysis and Design Language), 2005. LAAS-CNRS.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat und Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–53, 2008.

A pattern-based requirement specification language: Mapping automotive specific timing requirements*

Philipp Reinkemeier[1], Ingo Stierand[2], Philip Rehkop[1], and Stefan Henkler[1]

[1]philipp.reinkemeier,philip.rehkop,stefan.henkler@offis.de
[2]stierand@informatik.uni-oldenburg.de

Abstract: Today most requirement specifications are documents written in natural language. Natural language however is abiguous. Thus computer-aided verification of system-models against such requirement specifications is generally impossible. In this paper we propose a textual requirement specification language (RSL), that is based on patterns, which have a formally defined semantics. RSL is able to express requirements from multiple aspects (e.g. real-time, safety, etc.) on a system. In order to apply RSL in the domain of automotive systems, it has to support timing requirements as the *Timing Augmented Description Language* (TADL). TADL is the comming standard for handling timing information in the automotive domain. In this paper, we present a mapping of TADL constraints to RSL patterns.

1 Introduction

Stating requirements is an essential tasks in developing products. They build an interface between different groups of people, i.e. customers, engineers, project managers. Typically requirements are stated as natural language text. The disadvantage is that ambiguities can cause a huge amount of requirement changes in all development phases. This leads to higher development costs and may also be unacceptable if the product is deployed in safety critical environments. To avoid ambiguities, often formal languages are used to state requirements with well defined meaning. It however does not only require some training to write requirements in such formal languages, it may also be hard to read them.

The pattern based *Requirement Specification Language* RSL [Par10] bridges the gap by providing a formal language with well defined semantics that is still readable like natural language. RSL patterns consist of static text elements and attributes being instantiated by the requirements engineer. Each pattern has well defined semantics exploitable in verification tools and exhibits a natural language style in order to facilitate consistent interpretation of the written system specification across all project participants. Moreover, the underlying semantics of RSL establishes formal verification in order to ensure correctness of systems.

*This work was supported by the German Research Council (DFG) in SPES 2020 as well as by the ARTEMIS Joint Undertaking CESAR.

Writing requirements shall however be possible in an intuitive way. RSL allows writing of natural sounding requirements while being expressive enough to formalize complex requirements. To gain a high degree of intuition, the language consist of only a few constructs that can be easily remembered, allowing the user to fully understand the RSL and to choose the right pattern that fits the intended system properties. RSL aims at providing expression of a large scale of requirement domains. It defines patterns for each of the following categories:

- *Functional* patterns express requirements such as relationship between events, handling of conditions and invariants, and optional intervals in which a requirement (or parts of it) is valid.
- *Probability* patterns are used to specify the probabilities of failures or hazards occuring in safety critical systems.
- *Safety* related patterns outline relationships between system components. These patterns enable the specification of single point of failures or other failure and hazard dependencies between different components.
- *Timing* patterns are used to describe real-time behavior of systems. This includes recurring activations, jitter and delay.
- *Architectural* patterns specify the existence of architectural elements like components, events and connections between components.

There are already well established requirements specification languages for various aspects and application domains. In order to use RSL as a universal specification language it must be able to cover expressive power of those more specific approaches. In the automotive domain for example, there are mainly two relevant frameworks to specify timing requirements, namely the Timing Augmented Description Language (TADL) [Par09] and the AUTOSAR Timing Extensions (ARTE) [GbR09]. In this paper, we will justify about expressive power of RSL in comparison to TADL.

Though not illustrated in this paper, there already exists a compiler for translating instances of the patterns into observer automata. So verification can be done with state of the art model-checking tools like UPPAAL [BDL04].

Related Work The idea of requirement specification languages featuring a natural language style is not new. For instance in [HJD04] *boilerplates* for requirement specification are proposed. While this approach is also based on templates of natural language, there is not necessarily defined a formal semantics for them. Approaches based on controlled natural languages like *Attempto Controlled English* (ACE), first described in [FS96], or *PENG* [Sch04] are interesting as they come very close to natural language style and can be translated into a first-order logic system. However quantitative specifications like time intervals are inevitably for real-time systems. That cannot be done using such approaches because they usually do not provide a typed system. Graphical specification languages also provide an intuitive way to define requirements. Popular representatives are Live Sequence Charts (LCS) [DH01] and Timing Diagrams [DL02]. The mentioned approaches and formal language style languages like RSL have in common compositional techniques to define automaton semantics important for observer-based verification. RSL itself is

based on the *Contract Specification Language* (CSL) [GBC+08] defined in the SPEEDS project. RSL has been extended and evolved in that it is more consistent and modular.

Section 2 introduces the common concepts of the RSL and defines its semantics as far as relevant for this paper. Section 3 revisits the semantics of TADL constraints and discusses their covering by respective RSL patterns. Section 4 concludes the paper.

2 The Requirement Specification Language

The notation of RSL patterns normally contain optional parts that are not necessarily instantiated. A functional pattern for example describing causality between two events looks like this: **whenever** $event_1$ **occurs** $event_2$ **occurs [during** *interval***]**

Phrases in square brackets are optional, bold elements are static keywords of the pattern, and elements printed in italics represent attributes that have to be instantiated by the requirements engineer. The names of the attributes can be used later to be mapped to a given architecture, or they can be used to generate the architecture.

When specifying requirements there is no strict typing like *int* or *real* but there are categories that specify what is described by the different attributes. To not complicate the language too much and to be also expressive enough, the set of supported attributes has to be carefully chosen:

Events represent activities in the system. An event can be for example a signal, a user input by a pressed button, or a computational result. Events occur at distinct time instants and have no duration.

Conditions are logic and/or arithmetic expressions over variables and the status of events. Conditions can be used in two different ways. Firstly, they can be used as pseudo events to trigger an action if the condition becomes true. And secondly, they may represent the system state that has to hold for a specified time.

Intervals describe a continuous fragment of time whose boundaries are (relative) time measures, or events. They can be open "]x,y[", closed "[x,y]" or a combination thereof.

Components refer to entities that are able to handle events or condition variables. Typically, components refer to an architecture.

RSL supports various combinations of elements. It allows for example filtering of events by intervals or conditions. That is, wherever an event occurs in a pattern, one may also use *event* **during [** *interval* **]**, or *event* **under (***condition***)**. Thus, one can e.g. specify

whenever $request$ **during** $[activate, deactivate]$ **occurs** $response$ **occurs during** $[l, u]$

meaning that a response is constrained only if the $request$ happened after an $activate$ and before some $deactivate$ event. Filtering of events can be applied recursively. So $activate$ could again be filtered by some interval or condition.

In the following we define syntax and semantics of those RSL patterns that are used later to define the mapping to TADL-constraints. We define semantics in terms of *timed traces*

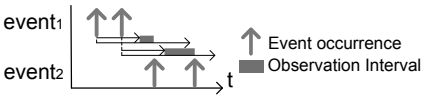


Figure 1: Semantics of pattern $F1$

and *timed languages* [AD94]. A *timed trace* over Σ is a pair (σ, τ), where $\sigma = \sigma_1\sigma_2\cdots$ is an infinite word over Σ, and $\tau = \tau_1\tau_2\cdots$ is a sequence of time instants $\tau_i \in \mathbb{R}$ and $\forall i : \tau_i \leq \tau_{i+1}$, at which the events σ_i occur. A *timed language* is a set of *timed traces* over Σ. The projection of (σ, τ) onto $\Sigma' \subseteq \Sigma$, written $(\sigma, \tau)_{|\Sigma'}$ is formed by deleting all $\sigma_i \notin \Sigma'$ along with their associated time value τ_i from (σ, τ). Projection is naturally extended to languages over timed traces.

Pattern F1: **whenever** *event*$_1$ **occurs** *event*$_2$ **[does not] occur[s] [during** *interval***].**

The pattern defines that an $event_2$ shall (not) occur whenever an $event_1$ has been occurred in the system. An optional interval confines the instant in time for $event_2$. Thereby *interval* may be defined either by other events like **[***startEvent, endEvent***]**, or by *timed values* forming the boundaries of an interval like **[**$0ms, 5ms$**]**. Timed values can also be negative, thereby allowing "backward specification". For mapping TADL constraints only boundaries in terms of timed values are needed.

Throughout this paper we make use of the following notations. We assume a set of events Σ_{Sys} that can be produced by a given system. The events occurring in an RSL pattern p are denoted Σ_p. For example, for the F1 pattern instance

whenever s **occurs** r **occur during [**l,u**]**.

we get $\Sigma_{F1} = \{s, r\} \subseteq \Sigma_{Sys}$. We further assume l and u to be timed values $l, u \in \mathbb{R}$. For $0 \leq l \leq u$ the semantics of pattern $F1$ is defined by the timed language $\mathcal{L}_{F1}$ over Σ_{F1}:

$$\mathcal{L}_{F1} = \{(\sigma, \tau) | \forall i \in \mathbb{N}, \exists j \in \mathbb{N} : \sigma_i = s \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)\}$$

The semantics is shown in Figure 1. Multiple activations of the pattern may run in parallel. The respective definition for $u \leq l < 0$ is symmetric.

The semantics of some RSL patterns can be modified by additional attributes. A pattern modified by the attribute **once** gets a strict interpretation of the action during the observed interval. The attribute **once** modifies $\mathcal{L}_{F1}$ by replacing the quantified $\exists j \in \mathbb{N}$ with $\exists! j \in \mathbb{N}$. Note that the attribute **once** can only be used in combination with an interval as otherwise the pattern would not terminate. Thus, the part "**during** *interval*" of the pattern becomes required. The optional part **does not** of the pattern modifies $\mathcal{L}_{F1}$ by replacing the quantified $\exists j \in \mathbb{N}$ by $\nexists j \in \mathbb{N}$ in the language definitions above.

Pattern R1: *event* **occurs sporadic with minperiod** $period_1$ **[and maxperiod** $period_2$**] [and jitter** *jitter***].**

The pattern defines that $event$ shall occur sporadically, relatively to some reference event which occurs with a minimum inter-arrival time $period_1$. An optional $jitter$ specifies the maximum amount of time each occurrence of $event$ can be delayed relatively to the

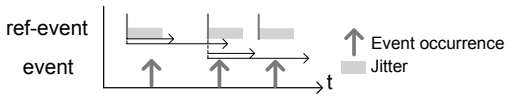


Figure 2: Semantics of pattern $R1$

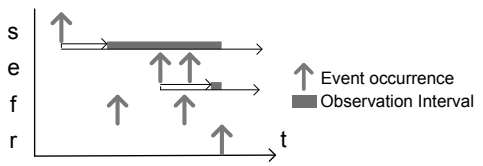


Figure 3: Semantics of sequences of events

occurrence of the reference event. If no jitter is defined it is assumed to be 0. The optional $period_2$ bounds the maximum inter-arrival time for subsequent occurrences of the reference event. For the $R1$ pattern instance

e **occurs sporadic with minperiod** T_1 **and maxperiod** T_2 **and jitter** J.

we have $\Sigma_{R1} = \{e\} \subseteq \Sigma_{Sys}$, and $T_1, T_2, J \in \mathbb{R}, 0 \leq T_1 \leq T_2, 0 \leq J$ are normalized time values. The language $\mathcal{L}_{R1}$ of the pattern is defined as follows:

$$\mathcal{L}_{R1} = \{(\sigma, \tau) | \exists \tau', \forall i \in \mathbb{N} : (\tau'_i \leq \tau_i \leq \tau'_i + J) \wedge (T_1 \leq \tau'_{i+1} - \tau'_i \leq T_2)\}$$

where T_1, T_2 form the lower and upper bound for the distance between subsequent occurrences of the reference event described by the *time sequence* $\tau' = \tau'_1 \tau'_2 \cdots$.

Sequences and Sets of Events The RSL language also provides specification of complex event occurrences by means of event sequences, event sets and logical operations on events. Event sequences are specified by an **and then** expression: **(***event* **and then** *event* **[during** *interval***])**

It can be used wherever the syntax allows to specify an event. They can be nested allowing for specification of event sequences longer than two. An optional interval in the expression confines the minimum and maximum distance between the involved events. In order to capture the semantics of event sequences, we denote aT the sequence of symbols specified by an **and then**-expression, and aT_i denotes the ith event in the sequence. In the following we restrict to the case where only the last event may be bounded by an additional interval. For example, the semantics of the $F1$ pattern instantiation

whenever s **occurs** (e **and then** f **and then** r **during** $[l', u']$) **occur during [**l,u**]**.

is depicted in Figure 3. Here, $\Sigma_{F1} = \{s, e, f, r\} \subseteq \Sigma_{Sys}$ denotes the set of events of the pattern, $l, u, l', u' \in \mathbb{R}$, and $aT = efr$. The language $\mathcal{L}_{F1_{aT}}$ is defined as follows:

$$\mathcal{L}_{F1_{aT}} = \{(\sigma, \tau) | \forall i \in \mathbb{N}, \exists \varrho = \varrho_1 ... \varrho_n, n = |aT| : \sigma_i = s \Rightarrow (i < \varrho_1) \wedge (\forall 1 \leq k \leq n : (\tau_i + l \leq \tau_{\varrho_k} \leq \tau_i + u) \wedge (\sigma_{\varrho_k} = aT_k) \wedge (\tau_{\varrho_1} + l' \leq \tau_{\varrho_n} \leq \tau_{\varrho_1} + u'))\}$$

Similarly to $\mathcal{L}_{F1}$ the attribute **once** may be applied or the optional part **does not** may be used, which replaces $\exists\varrho$ by $\exists!\varrho$, respectively $\nexists\varrho$.

Additionally, RSL provides a shortcut for the specification of N subsequent occurrences of the same event. The syntax is: *N* **times** *event*.

Further it is sometimes useful to define sets of events where ordering does not matter. Arbitrary ordering of a set $\{a, b, c\}$ of events can thus be expressed as

(a **and then** b **and then** c **during** $[l', u']$) **OR** (a **and then** c **and then** b **during** $[l', u']$)
OR (b **and then** a **and then** c **during** $[l', u']$) **OR** ...

As an abbreviation RSL provides the construct **set**$\{event, \cdots, event\}$ **[during** $interval$**]**.

Satisfaction-relation While each RSL pattern instantiation defines a distinct language, we have to reason about satisfaction of such constraint by a given system. So we assume system behaviour also be defined in terms of languages over timed traces. That is, a system S is defined by a language $\mathcal{L}_{Sys}$ over the alphabet Σ_{Sys}. S satisfies an RSL constraint p over an alphabet Σ_p if and only if $\mathcal{L}_{Sys|\Sigma_p} \subseteq \mathcal{L}_p$.

For multiple RSL-patterns $p_1 \cdots p_n$, each of which forms a *timed language* $\mathcal{L}_{p_i}$ over an alphabet Σ_{p_i}, the system obviously must satisfy all those constraints. This is equivalent to constructing the parallel composition of the constraints $\|_i\, p_i$ and then to check the system against it. The language of the parallel composition however is simply defined as follows:

$$\mathcal{L}_{\|} = \{(\sigma, \tau) | \forall i : (\sigma, \tau)_{|\Sigma_{p_i}} \in \mathcal{L}_{p_i}\}$$

If all alphabets Σ_{p_i} are the same, then $\mathcal{L}_{\|}$ is simply the intersection of all $\mathcal{L}_{p_i}$.

3 Mapping of TADL Constraints

The TADL semantics [Par09] of constraints defines how assumptions about occurrences of *events* must relate to each other for a set of timing constraints to be satisfied. That is the semantics of a constraint is given by its satisfaction-relation. The semantics of an *event* is the same as in RSL. For the occurrence of an event e TADL defines a function $dynamics(e) = \langle t_1, t_2, \cdots \rangle$, which associates that event with a list of strictly increasing time values. Thus it is a projection $(\sigma, \tau)_{|\{e\}}$ followed by a projection on the second component, where $\{e\} \subseteq \Sigma_{Sys}$. What remains is a strictly increasing *time sequence* $\tau = \tau_1 \tau_2 \cdots$ denoting the times of occurrences for event e in an observation.

An event chain relates two sets of events, *stimulus* and *response*. There is an intuition behind an event chain. However in [Par09] the semantics are left undefined and it is argued, that arising questions about causality would involve the semantics of the structural model, which was not intended. That means the causality of events is NOT defined by such an event chain.

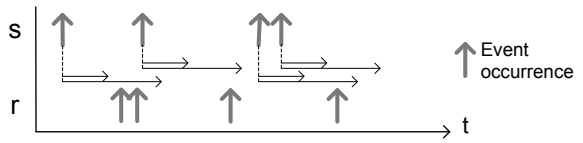


Figure 4: occurrences of s and r and the reference window for reaction constraint

3.1 Delay constraints

Delay constraints are applied on the *stimulus*- and *response*-sets of event chains and describe how the occurrences of events of that sets must relate. The parameters of each kind of delay constraint are given by a subset the following elements: A set S of events acting as stimuli, a set R of events acting as responses, a time offset l indicating the near edge of a time window, a time offset u indicating the far edge, and the size w of a sliding window.

Reaction constraint From [Par09]: A reaction time constraint c is satisfied for some given event behaviour if and only if for all events s in S, for all times t in $dynamics(s)$, for all events r in R, there exists at least one time v in $dynamics(r)$ such that $t+l \leq v \leq t+u$.

Using the previously defined interpretation of the function $dynamics(e)$, we can derive a timed language characterizing a reaction constraint. A reaction constraint forms a timed language $\mathcal{L}_{react}$ over an alphabet $\Sigma_{react} = \{S \cup R\} \subseteq \Sigma_{Sys}$, defined as

$$\mathcal{L}_{react} = \{(\sigma,\tau) | \forall i \in \mathbb{N}, \forall s \in S, \forall r \in R, \exists j \in \mathbb{N} : (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)\}$$

Characterizing the timed language of each TADL constraint is straight forward. For the remaining constraints we will thus not cite the TADL semantics and directly give the timed language. The semantics of a reaction constraint $\mathcal{L}_{react}$ can be expressed in RSL by a $F1$ pattern for each event $r \in \{r_1, \cdots, r_n\} = R$. The lower and upper bound l and u, respectively, are the same for all these $F1$ patterns. Thus, $\forall r \in R$:

whenever s_1 **OR** $\cdots$ **OR** s_m **occurs** r **occur during [**l,u**]**.

Proposition 1 $\mathcal{L}_{react} = \|_{r \in R}\, \mathcal{L}_{F1_r}$

Output Synchronization constraint An output synchronization constraint is a reaction constraint extended by the additional parameter w. It forms a *timed language* $\mathcal{L}_{output}$ over an alphabet $\Sigma_{output} = \{S \cup R\} \subseteq \Sigma_{Sys}$, defined as

$$\mathcal{L}_{output} = \{(\sigma,\tau) | \forall i \in \mathbb{N}, \forall s \in S, \exists x \in \mathbb{R}, \forall r \in R, \exists! j \in \mathbb{N} : (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r) \wedge (x \leq \tau_j \leq x + w)\}$$

The semantics of an output synchronization constraint $\mathcal{L}_{output}$ can be expressed in RSL by a $F1$ pattern for each event $r \in \{r_1, \cdots, r_n\} = R$. The lower and upper bound l and u, respectively, are the same for all these $F1$ patterns. Thus, $\forall r \in R$:

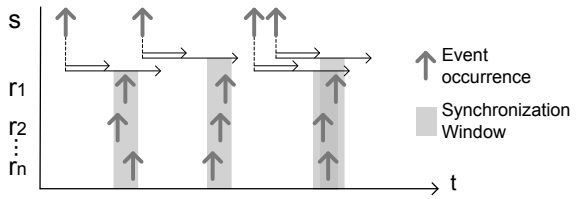


Figure 5: output synchronization constraint and the reference- and sliding-window

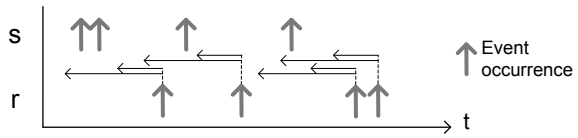


Figure 6: occurrences of s and r and the reference window for age constraint

whenever s_1 **OR** $\cdots$ **OR** s_m **occurs** r **occur during [**l,u**], once**.

Finally another $F1$ pattern of the following definition must be instantiated:

whenever s_1 **OR** $\cdots$ **OR** s_m **occurs set**$\{r_1, \cdots, r_n\}$ **during** $[0, w]$ **occur during [**l,u**]**.

Proposition 2 $\mathcal{L}_{output} = (\|_{r \in R}\, \mathcal{L}_{F1_{once_r}}) \parallel \mathcal{L}_{F1_{set_w}}$

Age constraint An age constraint forms a *timed language* $\mathcal{L}_{age}$ over an alphabet $\Sigma_{age} = \{S \cup R\} \subseteq \Sigma_{Sys}$, defined as

$$\mathcal{L}_{age} = \{(\sigma, \tau) | \forall i \in \mathbb{N}, \forall r \in R, \forall s \in S, \exists j \in \mathbb{N} : (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s)\}$$

The semantics of an age constraint $\mathcal{L}_{age}$ can be expressed in RSL by a $F1$ pattern for each event $s \in \{s_1, \cdots, s_n\} = S$. The lower and upper bound l and u, respectively, are the same for all these $F1$ patterns. Thus, $\forall s \in S$:

whenever r_1 **OR** $\cdots$ **OR** r_m **occurs** s **occur during [**$-u$,$-l$**]**.

Proposition 3 $\mathcal{L}_{age} = \|_{s \in S}\, \mathcal{L}_{F1_s}$

Input Synchronization constraint An input synchronization constraint is an age constraint extended by the additional parameter w. It forms a *timed language* $\mathcal{L}_{input}$ over an alphabet $\Sigma_{input} = \{S \cup R\} \subseteq \Sigma_{Sys}$, defined as

$$\mathcal{L}_{input} = \{(\sigma, \tau) | \forall i \in \mathbb{N}, \forall r \in R, \exists x \in \mathbb{R}, \forall s \in S, \exists! j \in \mathbb{N} : (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s) \wedge (x \leq \tau_j \leq x + w)\}$$

The semantics of an input synchronization constraint $\mathcal{L}_{input}$ can be expressed in RSL by a $F1$ pattern for each event $s \in \{s_1, ..., s_n\} = S$. The lower and upper bound l and u, respectively, are the same for all theses $F1$ patterns. Thus, $\forall s \in S$:

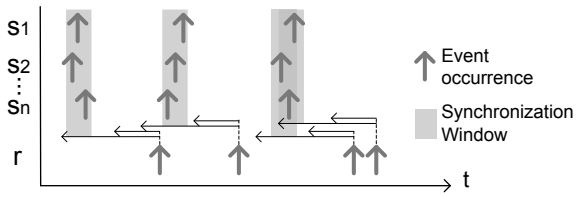


Figure 7: input synchronization constraint and the reference- and sliding-window

whenever r_1 **OR** ... **OR** r_m **occurs** s **occur during [**$-u$,$-l$**], once**.
Finally another $F1$ pattern of the following definition must be instantiated:

whenever r_1 **OR** ... **OR** r_m **occurs set**$\{s_1, ..., s_n\}$ **during** $[0, w]$ **occur during [**$-u$,$-l$**].**

Proposition 4 $\mathcal{L}_{input} = (\|_{s \in S}\ \mathcal{L}_{F1_{once_s}}) \parallel \mathcal{L}_{F1_{set_w}}$

3.2 Repetition rate constraints

Repetition rate constraints place restrictions on the distribution of the occurrences of a single event, forming a so called event model. TADL includes four kinds of repetition rate constraints, namely: *periodic*, *sporadic*, *pattern* and *arbitrary*. An additional *generic repetition rate constraint* is defined, that constitutes a semantic foundation for those repetition rate constraints. However it can **NOT** be instantiated directly.

Generic repetition rate constraint A generic repetition rate constraint is parametrized by the following elements: The event e, whose occurrences are constrained, a lower bound l on the time-distance between occurrences of e, an upper bound u on the time-distance between occurrences of e, the deviation J from an ideal point in time, at which the event e is expected to occur and the actual time it occurs and a count SP indicating whether it is subsequent occurrences or occurrences farther apart that are constrained.

A generic repetition rate constraint forms a *timed language* $\mathcal{L}_{generic}$ over an alphabet $\Sigma_{generic} = \{e\} \subseteq \Sigma_{Sys}$, defined as

$$\mathcal{L}_{generic} = \{(\sigma, \tau) | \exists \tau', \forall i \in \mathbb{N} : (\tau'_i \leq \tau_i \leq \tau'_i + J) \wedge (l \leq \tau'_{i+SP} - \tau'_i \leq u)\}$$

where $\tau' = \tau'_1 \tau'_2 ...$ is a *time sequence* of ideal times of occurrences for event e.

The semantics of the four kinds of repetition rate constraints are defined in terms of multiple *generic repetition rate constraints* and *reaction time constraints*. The definition of *periodic*, *sporadic* and *pattern* are based on generic repetition rate constraints with $SP = 1$. That semantics can be expressed in RSL by instantiating the following $R1$ pattern:

e **occurs sporadic with minperiod** l **and maxperiod** u **and jitter** J.
If we have a generic repetition rate constraint c, with $SP = 1$, $\mathcal{L}_c = \mathcal{L}_{R1}$ is given by the definitions of both languages.

The semantics of an *arbitrary repetition constraint* is based on generic repetition rate constraints with $SP \geq 1$ and $J = 0$. It can be expressed by the following $F1$ patterns:

whenever e **occurs** SP **times** e **occur during [**0,u**], once**.
whenever e **occurs** SP **times** e **does not occur during [**$0, l$**[.**

Proposition 5 *For a generic repetition rate constraint* c *with* $jitter(c) = 0$*:* $\mathcal{L}_c = \mathcal{L}_{F1_{aT,once}} \parallel \mathcal{L}_{F1_{aT,neg}}$

4 Conclusion

We presented a pattern-based textual requirements specification language (RSL), that exhibits a natural language style. The RSL allows writing of natural sounding requirements while providing a well defined formal semantics. Thus RSL can narrow the gap from informal requirement specification documents to formal specifications and verification techniques. The language provides patterns for various aspects like safety, functional and timing requirements. With regard to timing aspects we showed that the expressiveness of RSL is powerful enough to cover all constraints of the Timing Augmented Description Language (TADL). This enables using RSL to express a relevant set of timing requirements as defined in a well established specification language, while maintaining convenience and intuitiveness of natural language.

References

[AD94] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theor. Comp. Sc.*, 126(2), 1994.

[BDL04] G. Behrmann, A. David, and K. G. Larsen. A Tutorial on Uppaal. In *LNCS, Formal Methods for the Design of Real-Time Systems*, pages 200–237. Springer, 2004.

[DH01] Werner Damm and David Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, July 2001.

[DL02] H. Dierks and M. Lettrari. Constructing Test Automata from Graphical Real-Time Requirements. In *Proc. FTRTFT '02*, 2002.

[FS96] N. E. Fuchs and R. Schwitter. Attempto Controlled English (ACE). In *CLAW '96: First International Workshop on Controlled Language Applications*, pages 124–136, 1996.

[GBC+08] V. Gafni, A. Benveniste, B. Caillaud, S. Graph, and B. Josko. Contract Specification Language (CSL). Technical report, SPEEDS Consortium, April 2008.

[GbR09] AUTOSAR GbR. *Specification of Timing Extensions*, r4.0 rev1 edition, November 2009.

[HJD04] E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. Springer, 2 edition, 2004.

[Par09] Project TIMMO: TIMMO Partners. TADL: Timing Augmented Description Language version 2. Deliverable d6, The TIMMO Consortium, October 2009.

[Par10] Project CESAR: CESAR Partners. *RSL Reference Manual*. CESAR Consortium, 1.1 edition, 2010. Not publically available yet.

[Sch04] R. Schwitter. Representing Knowledge in Controlled Natural Language: A Case Study. In *Knowledge-Based Intelligent Information and Engineering Systems*, LNCS. 2004.

Herausforderungen für das Requirements Engineering in der Domäne Energie[1]

Friedrich-W. Fasse

RWE Consulting GmbH
Lysegang 11
45139 Essen

Kim Lauenroth

paluno - The Ruhr Institute for
Software Technology
Universität Duisburg-Essen
Gerlingstraße 16, 45127 Essen
kim.lauenroth@paluno.uni-due.de

Abstract: Der Ausbau derzeitiger Stromnetze zum intelligenten Stromnetz der Zukunft („Smart Grid“) stellt für die Domäne Energie einen massiven Umbruch dar. Bei der Realisierung des Smart Grid spielen Eingebettete Systeme eine zentrale Rolle. Zum einen werden sie zukünftig wesentliche Aufgaben der Steuerung von Energieerzeugung und -verbrauch übernehmen.

Zum anderen kommt ihnen eine bedeutsame Rolle bei der Interaktion zwischen den am Smart Grid beteiligten Einheiten zu. Nicht zuletzt ermöglicht erst der Einsatz leistungsfähiger Eingebetteter Systeme die Ausschöpfung vielfältiger Geschäftspotenziale mit neuen Produkten und Dienstleistungen sowie innovativen Geschäftsmodellen.

Bisher liegen nur wenige Erfahrungen bei der Entwicklung von Systemen mit vergleichbarer Komplexität vor. In Folge dessen müssen bereits initiale Analyseschritte im Requirements Engineering solcher Systeme mit besonderer Sorgfalt durchgeführt werden, um möglichst präzise Anforderungen für die weiteren Entwicklungsstufen dieser Systeme formulieren zu können.

Der vorliegende Beitrag gibt einen Überblick über aktuelle Entwicklungen in der Domäne Energie hinsichtlich des Smart Grid und leitet daraus zukünftige Herausforderungen für das Requirements Engineering ab.

1 Einleitung

Eingebettete Systeme sind schon heute ein wesentlicher Bestandteil von technischen Systemen in der Energiedomäne und haben bisher typischer Weise klar definierte und abgegrenzte Aufgaben z. B. bei der Steuerung und Regelung von Kraftwerken oder in Energienetzen übernommen.

Die aktuellen Entwicklungen in der Domäne Energie lassen jedoch darauf schließen, dass die Anforderungen an die Leistungsfähigkeit von Eingebetteten Systemen im energiewirtschaftlichen Kontext zukünftig deutlich steigen werden.

[1] Diese Arbeit wurde teilweise gefördert durch das Projekt SPES2020, gefördert durch das Bundesministerium für Bildung und Forschung (BMBF), Förderkennzeichen 01IS08045(L|V).

1.1 Entwicklung der Domäne Energie zum Smart Grid

Seit einigen Jahren zeichnet sich in der Energiedomäne ein massiver Umbruch hin zum „Intelligenten Stromnetz“ oder „Smart Grid“ ab. Der Begriff des Smart Grid [RWE10] steht für eine Aufrüstung des bestehenden Netzes, um den steigenden Herausforderungen an eine auch in Zukunft sichere und effiziente Stromversorgung gerecht zu werden.

Die Entwicklung wird durch eine Reihe von politisch, ökonomisch und ökologisch motivierten Zielen getrieben, welche insbesondere auch unmittelbaren Einfluss auf die zur Steuerung des zukünftigen Stromnetzes eingesetzte Informations- und Kommunikationstechnologie haben werden. So wird zur Erfüllung von Klimaschutzzielen und einer damit verbundenen nachhaltigen und ressourcenschonenden Energieversorgung bereits seit einigen Jahren der Ausbau der Erneuerbaren Energien wie bspw. Windenergie, Photovoltaik und Biogas verstärkt vorangetrieben. Dies führt dazu, dass die Energieerzeugung neben den bestehenden Großkraftwerken zunehmend auch von kleineren dezentralen Einheiten übernommen wird. Darüber hinaus wird ein effizienterer Umgang mit Energie angestrebt. Dies erfordert zum einen eine höhere Transparenz beim Energieverbrauch und zum anderen einen optimierten Einsatz elektrischer Geräte.

Aus diesen Entwicklungen erwachsen neue Herausforderungen für Funktion und Einsatz Eingebetter Systeme, die zukünftig sowohl zur Steuerung als auch zur Information und Kommunikation im Smart Grid eingesetzt werden. Höhere Anforderungen an die Steuerungsfunktion Eingebetteter Systeme ergeben sich vor allem daraus, dass der Energiemarkt durch die zunehmende Einspeisung aus Erneuerbaren Energien erheblich volatiler wird und damit eine verstärkte Angleichung des Erzeugungs- und Abnahmeverhaltens erforderlich macht. Darüber hinaus sollen Eingebettete Systeme die gezielte Steuerung elektrischer Verbraucher ermöglichen, in dem bspw. ein Wäschetrockner in einem Haushalt erst zu einem Zeitpunkt aktiviert wird, zu dem der Strom besonders günstig ist.

Die Herausforderungen hinsichtlich der Informations- und Kommunikationsfunktion von Eingebetteten Systemen manifestieren sich unter anderem darin, dass der Energieverbrauch einzelner oder mehrerer Aggregate im Haushalt zukünftig über eine nahezu kontinuierliche Messung transparenter wird und damit Energieeinsparpotenziale weitaus besser ermittelt werden können, als dies heute der Fall ist. Des Weiteren eröffnen sich nicht nur für Energieversorger sondern auch für andere (neue) Anbieter bisher ungeahnte Möglichkeiten, neue Produkte im Markt zu platzieren oder mit innovativen Geschäftsmodellen zusätzliche Erlöspotenziale zu erschließen. So können Energieversorger bspw. zeitvariable Tarife anbieten.

Mittels Eingebetteter Systeme - z. B. als Smart Meter - erhalten die Verbraucher Preissignale für einen zeitlich günstigen Energieverbrauch und damit einen Anreiz, Energie dann zu verbrauchen, wenn diese besonders günstig ist. Wenn die Informations- und Kommunikationstechnologie darüber hinaus auch in der Lage ist, angeschlossene elektrische Verbraucher automatisch zu aktivieren oder zu deaktivieren, übernimmt diese wiederum auch eine wichtige Steuerungsfunktion und trägt zu dem gewünschten Ausgleich von Energieeinspeisung und -entnahme bei.

Nicht zuletzt ergeben sich durch die gezielte Nutzung der vielfältigen Möglichkeiten leistungsfähiger Eingebetteter Systeme neue Geschäftsmodelle. So können sich zukünftige Marktakteure bspw. durch intelligente Verknüpfung von Smart Meter- und Smart-Home-Technologien zusätzliche Geschäftspotenziale erschließen. Ebenso ist es denkbar, durch Ausschöpfung der umfangreichen Möglichkeiten zur Informationsgewinnung und -aufbereitung als eine Art „Informationsbroker“ im Energiemarkt aufzutreten und basierend auf den verfügbaren Informationen neue Dienstleistungen anzubieten.

Erst der Einsatz entsprechend leistungsfähiger Eingebetteter Systeme schafft die technischen Voraussetzungen, die zunehmend anspruchsvolleren Steuerungs-, Informations- und Kommunikationsaufgaben im Smart Grid zu realisieren und entsprechende Mehrwerte durch neue Geschäftsmodelle zu generieren. Damit tragen Eingebettete Systeme nicht zuletzt auch zur Umsetzung der ehrgeizigen politischen, ökonomischen und ökologischen Ziele bei.

1.2 Fokus dieses Beitrags: Requirements Engineering

Bereits jetzt zeichnet sich ab, dass das Smart Grid eine extreme Komplexität aufweisen wird, die mit anderen Systemen nur schwer vergleichbar ist. Bisher liegen nur wenige Erfahrungen bei der Entwicklung von Systemen mit vergleichbarer Komplexität vor. Daher müssen bereits initiale Analyseschritte im Requirements Engineering solcher Systeme mit besonderer Sorgfalt durchgeführt werden, um möglichst präzise und umfassende Anforderungen für die weiteren Entwicklungsschritte formulieren zu können.

Aufgrund dieser Komplexität und der geringen Erfahrungen auf diesem Gebiet untersucht dieser Beitrag wesentliche Merkmale des Smart Grid und leitet daraus Herausforderungen ab, die sich insbesondere für das Requirements Engineering ergeben.

2 Komplexität des Smart Grid

Die Komplexität des Smart Grid ergibt sich im Wesentlichen aus drei Faktoren: Die hohe Anzahl an beteiligten Einheiten, das anfallende Informationsvolumen und die zunehmend dezentrale Energieerzeugung mit Erneuerbaren Energien.

2.1 Hohe Anzahl an beteiligten Einheiten

Betrachtet man eine typische Stadt mit 100.000 Haushalten, so ist in jedem Haushalt dieser Stadt ein konventioneller Stromzähler verbaut. Ein Smart Grid in dieser Stadt würde bedeuten, dass jeder dieser Stromzähler durch einen intelligenten Stromzähler und damit durch ein Eingebettetes System ersetzt würde. Daraus ergeben sich in dieser Stadt alleine in den Privathaushalten 100.000 Eingebettete Systeme. Die Stromzähler in den Gewerbe- und Industriebetrieben sind in diese Zahl noch nicht eingerechnet. Würde man diese Betrachtung auf ganz Deutschland erweitern, so ergäben sich noch wesentlich höhere Zahlen. Die Anzahl der Privathaushalte in Deutschland beläuft sich aktuell auf ca. 40 Mio. (Stand 2009 [St11]). Würde man in ganz Deutschland ein Smart Grid mit

intelligenten Stromzählern realisieren, wären damit auch 40 Mio. Eingebettete Systeme in einem solchen Netz aktiv. Mit dieser Größenordnung ist das Smart Grid von der Anzahl der beteiligten Einheiten vergleichbar mit dem Internet. Das Smart Grid weist allerdings zwei wesentliche Unterschiede zum Internet auf:

- Im Smart Grid findet primär eine 1:n Kommunikation statt, d.h. viele intelligente Stromzähler kommunizieren mit wenigen zentralen Einheiten der Energieversorger. Im Internet findet eine n:m Kommunikation statt, d.h. Einheiten kommunizieren untereinander.
- Im Smart Grid sind die beteiligten Einheiten im Wesentlichen permanent aktiv und kommunizieren mit den zentralen Einheiten. Im Internet sind nicht immer alle beteiligten Einheiten permanent aktiv, da die Internetnutzer nicht kontinuierlich im Netz aktiv sind und nicht alle Internetseiten durchgehend angefragt werden.

2.2 Informationsvolumen

Aus der zuvor beschriebenen Größenordnung der beteiligten Einheiten ergibt sich unmittelbar der zweite Komplexitätsfaktor: das Informationsvolumen. Jeder intelligente Stromzähler sammelt kontinuierlich Verbrauchsdaten, die in der Regel an die Energieversorger übertragen werden. Die aktuell in der Energiedomäne diskutierten Zeitintervalle der gemessenen Verbrauchsdaten bewegen sich im Bereich von wenigen Minuten.

Nimmt man für eine Beispielrechnung ein Intervall von 15 Minuten an, so würden pro Tag und pro Haushalt (4 x 24 =) 96 Messzeitpunkte für die Verbrauchsdaten anfallen. Hochgerechnet auf die in Abschnitt 2.1 angeführte Stadt würden pro Tag 9,6 Mio. Messzeitpunkte anfallen. Bei einer Größe von 1 Kilobyte pro Datensatz ergäben sich daraus ca. 10 Gigabyte Daten pro Tag. Dieses Datenvolumen muss durch Kommunikationsnetze übertragen und ausgewertet sowie an geeigneter Stelle (im Allgemeinen beim Energieversorger) für einen definierten Zeitraum gespeichert werden.

2.3 Erneuerbare Energien und dezentrale Energieerzeugung

Den Erneuerbaren Energien wie bspw. Windkraft und Photovoltaik wird in der Energiedomäne ein hohes Potenzial zugesprochen [UE11]. Windkraft bzw. Photovoltaik sind jedoch von der verfügbaren Menge an Wind resp. der Sonneneinstrahlung abhängig. Wenn der Anteil dieser Energiequellen an der Energieversorgung zunimmt, wächst automatisch der Einfluss aktueller Wetterbedingungen auf die Energieversorgung. Damit steigt gleichzeitig die Volatilität im Netz.

Daher müssen unvorhersehbare Wetterereignisse (bspw. Regenwetter oder starker Wind) bei der Planung der Energieversorgung berücksichtigt werden und ausreichend Reservekapazitäten (bspw. Speicherkraftwerke) oder schnell zu-/ abschaltbare Lasten vorgehalten werden, um nicht prognostizierte Ausfälle oder kurzfristig entstehende Überkapazitäten bei den Erneuerbaren Energien zu kompensieren. Neben den Erneuerbaren Energiequellen aus Wind und Sonne stehen mit Biogasanlagen und Blockheizkraftwerken weitere zahlreiche kleinere Energiequellen zur Verfügung, die ihren Beitrag zur gesamten

Energieversorgung leisten. Die Steuerung solcher kleineren Anlagen birgt jedoch Probleme:

- *Energiequellen sind teilweise nicht schaltbar*: Erneuerbare Energiequellen, wie bspw. Photovoltaikanlagen liefern Energie, wenn die Sonne scheint. Wenn die Sonne nicht scheint, wird keine Energie geliefert. Des Weiteren ist die Produktion elektrischer Energie unter Umständen ein Abfallprodukt (bspw. in Blockheizkraftwerken (BHKW)), und ein Abschalten der Energiequelle ist nicht möglich, da die Lieferung des Hauptprodukts des Kraftwerkes (bspw. Wärme) für eine dritte Partei notwendig ist (Wärme geführtes BHKW).
- *Eigentümer verfolgen Gewinnmaximierung*: Viele Quellen von Erneuerbaren Energien sind nicht im Eigentum der Energieversorger. Die Photovoltaikanlage auf dem Dach gehört bspw. dem Hausbesitzer oder das Blockheizkraftwerk auf dem Gelände einer Fabrik gehört dem Fabrikbesitzer. Die Vergütung der Eigentümer dieser Anlagen erfolgt typischer Weise nach der abgenommenen oder eingespeisten Energiemenge. Damit geht in der Regel das Interesse des Eigentümers einher, dass die Anlage möglichst kontinuierlich Energie liefert und nicht abgeschaltet wird. Zudem verpflichtet das Erneuerbare-Energien-Gesetz (EEG) den Netzbetreiber zu einer vorrangigen Abnahme von Energiemengen aus EEG-Erzeugung
- *Ausgleich von unkontrolliertem Ausfall von Energiequellen*: Erneuerbare Energiequellen können aus unterschiedlichen Gründen unvorhersehbar ausfallen, wenn bspw. bei einer Biogasanlage der Gasvorrat erschöpft ist oder wenn der Generator auf Grund mangelnder Wartung ausfällt. Ein effizientes Steuerungssystem muss in der Lage sein, solche ungeplanten Ereignisse durch geeignete Maßnahmen auszugleichen.

2.4 Herausforderungen für das Requirements Engineering

Für das Requirements Engineering ergeben sich aus der hohen Anzahl an beteiligten Einheiten sowie aus dem großen Informationsvolumen die folgenden Herausforderungen:

- *Analyse von Wechselwirkungen*: Auf Grund der hohen Anzahl beteiligter Einheiten können unerwartete Emergenzeffekte entstehen, die zu unerwünschtem Systemverhalten führen (bspw. Überlastung von Teilsystemen). Mögliche Folgen aus Emergenzeffekten müssen durch geeignete Requirements Engineering-Techniken erkannt und entsprechend berücksichtigt werden.
- *Berücksichtigung von technischen und ökonomischen Aspekten*: Das Beispiel der Steuerung auch kleinerer dezentraler Energieerzeuger/ -einspeiser zeigt, dass technische und ökonomische Interessen der beteiligten Partner einander widersprechen können. Techniken zur Ermittlung von Anforderungen müssen daher insbesondere die Integration von technischen und ökonomischen Interessen berücksichtigen.

- *Anforderungen an die Skalierbarkeit*: Requirements Engineering-Techniken müssen dediziert auf die Ermittlung von Skalierbarkeitsanforderungen abzielen, damit die entwickelten Eingebetteten Systeme in der notwendigen Größenordnung operieren können. Diese Arten von Anforderungen sind insbesondere für den Entwurf und die technische Auslegung von Systemen im Smart Grid notwendig.
- *Validierung von Anforderungen in Bezug auf die Skalierbarkeit*: In Bezug auf die Validierung von Anforderungen muss für Systeme im Smart Grid geprüft werden, ob definierte Anforderungen auch in Systemen erfüllt werden können, welche die zuvor beschriebene Größenordnung aufweisen. Als Beispiel für eine solche Anforderung kann das Zeitintervall herangezogen werden, in dem die Verbrauchsdaten an den Energieversorger übermittelt werden.

3 Rahmenbedingungen in der Domäne Energie

In der Domäne Energie sind zwei wesentliche Rahmenbedingungen zu beachten: Erstens ist das Energienetz ein massiv räumlich verteiltes System aus Einheiten, die zweitens eine lange Lebensdauer und hohe Zuverlässigkeit aufweisen müssen.

3.1 Massive räumliche Verteilung des Systems

Das bestehende Stromnetz zeichnet sich bereits jetzt durch eine massive räumliche Verteilung der beteiligten Einheiten aus. Diese Verteilung bezieht sich nicht nur auf Kraftwerke, dezentrale Einspeiser wie bspw. Photovoltaikanlagen, Umspannwerke und Verteilstationen sondern insbesondere auch auf die in der Fläche in großer Zahl vorhandenen Stromzähler.

Im zukünftigen Smart Grid ist davon auszugehen, dass die Anzahl beteiligter Einheiten und damit die massive räumliche Verteiltheit weiter zunehmen wird. Betrachtet man bspw. die Stromzähler, welche sukzessive durch Smart Meter ersetzt werden sollen, bedeutet dies, dass Wartungs- und Instandsetzungsarbeiten nach Installation aus betriebswirtschaftlichen Gründen kaum noch manuell vor Ort durch einen Techniker/ Dienstleister wahrgenommen werden können. Hieraus ergeben sich unter anderen die folgenden Konsequenzen:

- Die Installation der Eingebetteten Systeme muss mit maximaler Effizienz erfolgen, damit der Aufwand für die Realisierung eines Smart Grid überhaupt möglich ist.
- Eingebettete Systeme wie der intelligente Stromzähler müssen nach der Installation möglichst zuverlässig und störungsfrei arbeiten.
- Physikalische Anpassungen der Systeme durch einen Techniker/ Dienstleister kämen auf Grund des Aufwands und der Kosten kaum in Frage.
- Auch Änderungen der Software, bspw. Fehlerbehebungen oder das Ergänzen neuer Funktionalitäten müssen über geeignete Kommunikationsnetze und weitestgehend automatisch erfolgen.

3.2 Lebensdauer und Zuverlässigkeit der Systeme

Konventionelle Stromzähler, wie sie in deutschen Haushalten verbaut sind, haben eine typische Lebensdauer von 20 bis 30 Jahren innerhalb derer sie störungsfrei arbeiten. Die lange Lebensdauer ist nicht zuletzt auf Grund der in Abschnitt 3.1 erwähnten massiven räumlichen Verteiltheit ein wesentlicher Wirtschaftlichkeitsfaktor dieser Geräte. Überträgt man die Anforderung an die Lebensdauer (20 bis 30 Jahre) auf das Smart Grid und bspw. den intelligenten Stromzähler, ergeben sich hieraus unter anderem die folgenden Konsequenzen:

- Die technischen Bauteile müssen über einen Zeitraum von 20 bis 30 Jahren fehlerfrei funktionieren.
- Die physikalischen Schnittstellen zum Stromnetz sowie zum Informations- und Kommunikationsnetz (welches der Übertragung von Information zum Stromanbieter dient) können kaum/ gar nicht angepasst werden.
- Die Hardware-Ressourcen der Einheiten (CPU, Speicher, etc.) müssen so dimensioniert sein, dass sie nicht nur die Funktionalitäten von heute mit angemessener Performanz umsetzen können, sondern auch Funktionalitäten umsetzen können, die in 10 oder 20 Jahren erdacht und entwickelt werden.
- Die Software der Einheiten muss derart flexibel entworfen und entwickelt werden, dass auch zukünftig erdachte und entwickelte Funktionen in die bestehende Software integriert werden können.

3.3 Herausforderungen für das Requirements Engineering

Für das Requirements Engineering ergeben sich aus der massiven räumlichen Verteilung sowie der geforderten Lebensdauer und Zuverlässigkeit der Systeme die folgenden Herausforderungen:

- *Anforderungen zur Berücksichtigung der räumlichen Verteiltheit*: Bei der Ermittlung von Anforderungen muss berücksichtigt werden, dass die zu entwickelnden Systeme nach der Installation nur mit großem Aufwand durch einen Techniker/ Dienstleister erreicht und manuell modifiziert werden können.
- *Anforderungen für die nächsten 20 bis 30 Jahre*: Bei der Ermittlung von Anforderungen an Eingebettete Systeme im Smart Grid muss berücksichtigt werden, dass auch Anforderungen definiert werden, die zwar heute noch nicht zwingend relevant sind, aber innerhalb der nächsten 20 Jahre relevant werden könnten. Umgekehrt muss bei der Ermittlung die Frage gestellt werden, ob Anforderungen auch noch in 20 Jahren relevant sein werden.
- *Anforderungen an eine Lebensdauer von bis zu 30 Jahren*: Bei der Ermittlung von Anforderungen muss ebenfalls der Faktor berücksichtigt werden, dass Eingebettete Systeme die angestrebte Lebenserwartung von bis zu 30 Jahren erfüllen. Dieser Faktor bezieht sich sowohl auf die Hardware als auch auf die Software der Eingebetteten Systeme.

- *Validierung in Bezug auf eine lange Lebensdauer und auf Zuverlässigkeit*: Anforderungen an Eingebettete Systeme müssen dahingehend überprüft werden, ob diese Anforderungen einer langen Lebensdauer der Systeme entgegenstehen oder sich negativ auf die Zuverlässigkeit auswirken.

Insgesamt ist zu berücksichtigen, dass im Smart Grid neben dem eigentlichen Stromnetz ein zweites Netz aufgebaut wird - das Informations- und Kommunikationsnetz (IKT-Ring). Dies führt dazu, dass völlig neue Steuerungs- und Kontrollvorgänge stattfinden werden, welche das Stromnetz selbst nicht destabilisieren dürfen.

4 Datenschutz und Sicherheit im Smart Grid

Aus den technischen Möglichkeiten zur Gewinnung von Daten im Smart Grid ergibt sich unmittelbar die Frage nach dem Datenschutz und der Sicherheit bzw. dem Schutz vor Missbrauch.

4.1 Datenschutz

Die Informationen, die ein intelligenter Stromzähler an den Energieversorger überträgt, bestehen im Wesentlichen aus der aktuellen Uhrzeit und dem aktuellen Stromverbrauch. Es ist beabsichtigt, diese Informationen zukünftig mehrfach pro Stunde zu übertragen und für Abrechnungs- und ggf. weitere Zwecke sowie zur Erstellung von Lastprofilen zu nutzen (siehe Abschnitt 1.1). Basierend auf diesen Informationen lassen sich unter anderem folgende Aussagen über einen einzelnen Haushalt ableiten [Bi09]:

- *Transparenz über die Lebensweise in einem Haushalt*: Aus dem Lastprofil eines Haushalts kann abgeleitet werden, wann Nachtruhe in einem Haushalt herrscht. Das Lastprofil zu dieser Zeit bleibt für den Zeitraum der Nachtruhe auf einem konstant niedrigen Niveau, da nur die dauerhaft eingeschalteten Elektrogeräte (bspw. Kühl- oder Gefrierschrank) Energie verbrauchen.
- *Auskunft über die Nutzung von elektrischen Geräten*: Es kann bspw. abgeleitet werden, ob und wann im Haushalt (elektrisch) gekocht wird oder wann der Fernseher läuft. Die Benutzung eines Elektroherdes ist sehr deutlich im Lastprofil erkennbar, da ein Elektroherd im Betrieb eine sehr hohe Energieleistung benötigt.
- *Auskunft über Abwesenheitszeiten im Haushalt*: Ein langer Zeitraum an geringer Last über mehrere Tage hinweg deutet darauf hin, dass im Haushalt niemand anwesend ist. Analog deutet eine regelmäßig geringe Last tagsüber darauf hin, dass im Haushalt tagsüber niemand zu Hause ist.

4.2 Sicherheit

Aus den erhobenen Daten in Verbindung mit den technischen Möglichkeiten im Smart Grid erwachsen Risiken des Datenmissbrauchs [Fi10]. So sind unter anderem folgende Missbrauchsszenarien denkbar:

- *Lastprofil zeigt den günstigsten Zeitpunkt für eine Einbruch*: Wenn das Lastprofil eines Haushalts einem Dritten zugänglich gemacht wird, kann er daraus mögliche Zeitpunkte und Dauer einer Abwesenheitszeiten der Haushaltsmitglieder ableiten, um so den besten Zeitpunkt für einen Einbruch zu ermitteln.
- *Abschalten der Energie in einem Haushalt von außen*: Durch die geplanten Steuerungsmöglichkeiten von elektrischen Geräten (siehe Abschnitt 1.1) kann ein Dritter die Energie in einem Haushalt komplett abschalten und damit bspw. eine Überwachungsanlage in einem Haus außer Funktion setzen, um einen Einbruch zu erleichtern.
- *Sabotage der Stromversorgung in Teilnetzen*: Erlangt ein Dritter Zugang zu den Steuerungsmöglichkeiten intelligenter Stromzähler, bspw. in einer ganzen Stadt, so könnte das Energienetz dieser Stadt durch gezieltes Ein- und Ausschalten einer Vielzahl elektrischer Geräte überlastet und damit schlimmstenfalls die Energieversorgung dieser Stadt außer Funktion gesetzt werden.

4.3 Herausforderungen für das Requirements Engineering

Die angeführten Beispiele zeigen, dass Datenschutz und Sicherheit essenzielle Themen im Smart Grid sind. Zwar sind personenbezogene Daten zur Steuerung des Netzes nicht notwendig.

Vielmehr reichen pseudomysierte und kumulierte Werte aus, so dass ein direkter Rückschluss auf individuelle Lastgänge entfällt. Dennoch ergeben sich unter Datenschutz- und Sicherheitsaspekten unter anderen die folgenden Herausforderungen für das Requirements Engineering:

- *Datenschutzbedarf erkennen*: Techniken zur Erhebung von Anforderungen müssen insbesondere den Schutz sensibler Daten im Smart Grid gewährleisten, da intelligente Stromzähler über weitaus mehr Möglichkeiten der Datengewinnung und -aufbereitung verfügen als bislang eingesetzte Geräte.
- *Sicherheitsanforderungen identifizieren*: Requirements-Engineering-Techniken müssen darauf abzielen, Anforderungen zu erheben, die die Sicherheit im Smart Grid gewährleisten. Insbesondere muss darauf abgezielt werden, mögliche Missbrauchsszenarien frühzeitig zu identifizieren und Anforderungen zu entwickeln, um den Missbrauch von Smart-Grid-Technologien zu verhindern.
- *Missbrauchspotenzial minimieren*: Techniken zur Validierung von Anforderungen müssen darauf abzielen, ein mögliches Bedrohungspotenzial zu identifizieren, das von einer definierten Anforderung ausgeht, bspw. könnte geprüft werden, ob eine Datenanforderung im Smart Grid für Missbrauchszwecke genutzt werden kann oder ob eine funktionale Anforderung die Sicherheit im Smart Grid gefährdet.

5 Zusammenfassung

Die Aufrüstung der Energienetze in Richtung Smart Grid erfordert zunehmend den Einsatz leistungsstarker, umfassend vernetzter Eingebetteter Systeme für eine effiziente Nutzung zukünftiger Smart-Technologien. Eingebettete Systeme erfüllen dabei zum einen eine wesentliche Steuerungsfunktion zum Ausgleich von Energieerzeugung und -verbrauch. Zum anderen nehmen sie umfassende Aufgaben der Information und Kommunikation zwischen den im Smart Grid aktiven Einheiten wahr.

Der Beitrag fokussiert auf die zukünftigen Herausforderungen für das Requirements Engineering zur Entwicklung und Beherrschung Eingebetteter Systeme in der Domäne Energie. Zu den wesentlichen Herausforderungen, welche in diesem Beitrag diskutiert wurden, zählt die Komplexität des zukünftigen Smart Grid. Diese ist mit der Komplexität anderer Systeme nur schwer vergleichbar. Weitere Herausforderungen resultieren aus der massiven räumlichen Verteilung sowie den Anforderungen an Lebensdauer und Zuverlässigkeit zukünftiger Eingebetteter Systeme. Schließlich muss sich das Requirements Engineering den besonderen Herausforderungen an Datenschutz und Sicherheit im Smart Grid stellen.

Nicht zuletzt entscheidet die zielführende Umsetzung datenschutzrechtlicher sowie sicherheitstechnischer Aspekte bei der Entwicklung und Implementierung von Eingebetteten Systemen über Akzeptanz oder Reaktanz der „Smart"-Technologien und der damit einhergehenden vielfältigen Möglichkeiten neuer Produkte oder innovativer Geschäftsmodelle im Smart Grid.

6 Literaturverzeichnis

[Bi09] Biermann, K.: Smart Meter - Was der intelligente Stromzähler ausplaudern darf. ZEIT ONLINE, 14.09.2009, http://www.zeit.de/digital/datenschutz/2009-09/uld-smartmeter-gutachten.

[Fi10] Fischermann, T.: Intelligente Stromzähler - Attacke im Sicherungskasten. DIE ZEIT, 16.09.2010 Nr. 38, http://www.zeit.de/2010/38/Smart-Grid-Hacker.

[Na11] Webseite der Voyager-Sonden, http://voyager.jpl.nasa.gov/, abgerufen am 3.1.2011.

[RWE10] RWE-Broschüre: Wir sagen, was dahintersteckt – Begriffe, die im Zusammenhang mit der Energieversorgung von morgen häufig fallen, und ihre Bedeutung. http://www.rwe.com/web/cms/mediablob/de/496876/data/496884/1/factbook/Factbook.pdf.

[St11] Webseite „Statistische Ämter des Bundes und der Länder", http://www.statistik-portal.de/Statistik-Portal/de_jb01_jahrtab4.asp, Stand 3.8. 2010.

[Ue11] Webseite „Unendlich viel Energie", http://www.unendlich-viel-energie.de/, abgerufen am 3.1.2011.

Modellgetriebene Validierung von System-Architekturen

André Pflüger, Wolfgang Golubski

Westsächsische Hochschule Zwickau
Fachgruppe Informatik
Dr.-Friedrichs-Ring 2A
08056 Zwickau
{Andre.Pflueger|golubski}@fh-zwickau.de

Stefan Queins

SOPHIST GmbH
Vordere Cramergasse 13
90478 Nürnberg

Stefan.Queins@sophist.de

Abstract: Die System-Architektur bildet das Fundament eines jeden eingebetteten Systems. Der Architekt muss sie den sich ständig ändernden Bedingungen anpassen und die Validität gegenüber den System-Anforderungen nachweisen. Der vorgestellte Ansatz reduziert diese zeitaufwändige Arbeit durch einen automatisierbaren Validierungsprozess auf Basis der Unified Modeling Language (UML) und unterstützt den Architekten beim Architekturentwurf und der Impact-Analyse.

1. Motivation

Bei softwareintensiven, eingebetteten Systemen ist eine Projektlaufzeit über mehrere Jahre eher die Regel als eine Ausnahme. Die Entwicklung wird mit sich häufig ändernden Bedingungen konfrontiert, die eine kontinuierliche Planung und Entwicklung auf Basis vereinbarter Anforderungen fast unmöglich machen. Der Kunde verlangt vom Hersteller größtmögliche Flexibilität bei der Systementwicklung. Gleichzeitig steigt die Komplexität der zu entwickelnden Systeme kontinuierlich und damit auch die Anzahl und Beziehungen der Anforderungen untereinander. Mittlere bis hohe vierstellige Werte sind nicht ungewöhnlich.

Das Fundament eingebetteter Systeme bildet die System-Architektur. Sie wird beispielsweise durch die Faktoren

- funktionale und nicht-funktionale Anforderungen,
- eingesetzte Technik und Technologien,
- Verständnis des aktuellen Systems,
- Erfahrungen aus früheren Entwicklungen sowie
- dem Faktor Mensch

beeinflusst und ist für die Abschätzung der Entwicklungskosten und -zeit von Bedeutung. Es sind die Aufgaben des Architekten sie unter Berücksichtigung aller Einflussfaktoren zu entwerfen und deren Validität gegenüber den architekturrelevanten Anforderungen sicherzustellen. Diese Validierung ist ein arbeitsintensiver und zeitaufwendiger Vorgang, der bei jeder Änderung eines Einflussfaktors, z.B. einer Anforderung, wiederholt werden muss. Dabei sieht das Vorgehen des Architekten in vielen Fällen folgendermaßen aus:

Der Architekt erstellt einen ersten Architekturentwurf und verfeinert diesen iterativ. Für die Validierung wird die Methode des scharfen Hinsehens verwendet. Damit ist ein manueller Abgleich der Architektur mit den Anforderungen ohne ein definiertes Vorgehen gemeint, was zum unabsichtlichen Auslassen einiger Aspekte führen kann. Die Validierung beruht auf Abschätzungen, die auf Erfahrungen des Architekten basieren, und nur selten, meist bei kritischen Anforderungen, auf belegbaren Zahlen. Aufgrund des Zeit- und Arbeitsaufwands wird die Dokumentation und Validierung nur für bestimmte, beispielsweise Review-Termine vollständig durchgeführt.

Der in diesem Paper vorgestellte Ansatz definiert einen teilautomatisierbaren Validierungsprozess, der in das iterative Vorgehen zum Entwurf der System-Architektur eingebaut wird. Sogenannte Validierungsziele fassen architekturrelevante Anforderungen auf einem für den Architekturentwurf passenden Abstraktionsniveau zusammen, was den Überblick vereinfacht und die komplexen Anforderungsbeziehungen untereinander sichtbar werden lässt. Zu jedem Validierungsziel legt der Architekt ein Verfahren zur Feststellung der Validität eines Validierungsziels fest, das sogenannte Validierungsverfahren. Die Validierung der System-Architektur ist erfolgreich, falls alle Validierungsziele erfolgreich validiert werden. Der Ansatz setzt eine modellbasierte Dokumentation des Systems mit UML voraus. Für jedes Validierungsziel wird eine validierungszielspezifische Notation erstellt, mit der die für das zugehörige Validierungsverfahren benötigten Informationen im sogenannten Validierungsmodell modelliert werden. Die Modelldaten werden zum einen für die Validierung und zum anderen für eine lückenlose Dokumentation des Architektur-Entwurfs, dessen Entstehung und Anpassung genutzt. Änderungen zum Finden einer passenden System-Architektur werden direkt am Modell vorgenommen. Der Ansatz reduziert den Arbeits- und Zeitaufwand für die Validierung, indem möglichst viele Validierungsverfahren automatisiert durchgeführt und die dazugehörigen Werte direkt aus dem Validierungsmodell ausgelesen werden. Der Architekt wird beim Entwurf und der Impact-Analyse nach Anforderungsänderungen unterstützt und Änderungen an der Architektur können aufgrund der lückenlosen modellbasierten Dokumentation nachvollzogen werden.

Im weiteren Verlauf des Papers wird in Kapitel 2 zunächst auf verwandte Arbeiten verwiesen. Danach folgt in Kapitel 3 die Beschreibung eines Beispiels aus dem Radar-Bereich, das bei der Beschreibung des Validierungsprozesses in Kapitel 4 zur Veranschaulichung herangezogen wird. Kapitel 5 listet die Vorteile und die bisherigen Erfahrungen mit dem Ansatz auf, bevor Kapitel 6 ein Fazit zieht und einen Ausblick über zukünftig geplante Entwicklungen gibt.

2. Verwandte Arbeiten

Bei unseren Nachforschungen haben wir keinen Ansatz entdeckt, der einen modellgetriebenen Validierungsprozess für die System-Architektur gegenüber den System-Anforderungen definiert und den Architekten bei der Entwurfsarbeit und der Impact-Analyse unterstützt.

Das Tool PREEvision von aquintos [Aqu11] erlaubt die Modellierung, Entwicklung und Bewertung von Elektronik/Elektrik (E/E)-Architekturen. Für die Modellierung steht eine auf einem eigenen Metamodell basierende domänenspezifische grafische Notation und ein Datenmodell zur Eingabe der architekturrelevanten Informationen zur Verfügung. Der Hauptfokus liegt auf der Entwicklung und Validierung der E/E-Architektur, während sich unser Ansatz auf die Validierung der System-Architektur als Ganzes, Kombination von Soft- und Hardware, konzentriert. Statt eines eigenen domänenspezifischen Metamodells setzt unser Ansatz auf eine Erweiterung der domänenspezifischen Notation mit Hilfe des Profilmechanismus der UML, wodurch die Erweiterung vorhandener domänenspezifischer Notationen für die Validierung mit geringem Aufwand und validierungszielspezifisch möglich ist.

Für die Modellierung von System-Architekturen kann die System Modeling Language (SysML) [OMG10a], die Architecture Analysis & Design Language (AADL) [CMU11] oder das für eingebettete und Echtzeit-Systeme gedachte Modeling and Analysis of Real-Time and Embedded Systems (MARTE)-Profil [OMG10b] eingesetzt werden. Für die Erstellung von speziell auf die einzelnen Validierungsziele angepassten Notationen sind sie allerdings ungeeignet.

Das vom Aerospace Vehicle Systems Institute initiierte Projekt System Architecture Virtual Integration (SAVI) [Fei10] umfasst drei Schlüsselkonzepte: ein Referenzmodell, ein Modell-Repository mit -Bus sowie ein modellbasierter Entwicklungsprozess. Das Konzept ermöglicht auf das Referenzmodell abgebildete Modelle, z.B. auf Basis von UML, AADL oder SysML, mit in einem Repository hinterlegten Anforderungen zu vergleichen. Damit ließe sich zwar theoretisch eine Validierung der System-Architektur gegenüber Anforderungen durchführen, der Mehraufwand wäre aber sehr groß, da das Projekt eine deutlich andere Zielsetzung hat. Zudem existiert es bisher nur in der Beschreibung der Machbarkeitsstudie.

Als ein Beispiel für Ansätze zur Verifikation von System-Architekturen wird CoFluent Studio [CoF10] vorgestellt. Es bietet ein Werkzeug zum Testen der System-Architektur in einer virtuellen Umgebung an. Für die Dokumentation werden UML-Modelle inklusive der Profile SysML und MARTE eingesetzt. Diese Modelle werden in ausführbare SystemC-Transaction-Level-Modelle transformiert. Das Konzept ist vergleichbar zum Executable UML [Bal02], hat aber eine andere Zielsetzung gegenüber unserem Ansatz. CoFluent Studio verifiziert die modellierten Informationen durch eine Simulation in einer speziellen Umgebung. CoFluent Studio könnte als Validierungsfahren zum Testen eines Validierungsziels verwendet werden, es ermöglicht aber weder die Validierung der System-Architektur als Ganzes, noch unterstützt es den Architekten beim Entwurf oder der Impact-Analyse.

3. Beispiel für die Validierung einer System-Architektur

Die Beschreibung des Validierungsprozesses und der Begrifflichkeiten im folgenden Kapitel werden mit Hilfe eines stark vereinfachten Radar-Systems unterstützt. Die Funktionsweise wird anhand der in Abbildung 1 dargestellten Software-Sicht erläutert. Sie ist Teil der System-Architektur-Dokumentation im Entwicklungsmodell.

Die Reflektionen der vom Radar ausgestrahlten elektromagnetischen Wellen (Signale) werden über die Antenne empfangen und in vier Schritten zu den für die Anzeige auf dem Radarbildschirm benötigten Tracks verarbeitet. Während der Signalverarbeitung werden die empfangenen Reflektionen zunächst digitalisiert und zu sogenannten Beams zusammengefasst. Aus diesen werden Störungen herausgefiltert, so dass nur noch die von Objekten reflektierten Signale, sogenannte Hits, übrig bleiben. In der nächsten Verarbeitungsstufe, der Plotverarbeitung, werden alle zu einem Objekt gehörenden Hits zu Plots zusammengefasst bevor sie in der Trackverarbeitung mit weiteren z.B. historischen Daten verknüpft werden, um beispielsweise die Bewegung eines Objektes auf dem Radarbildschirm verfolgen zu können.

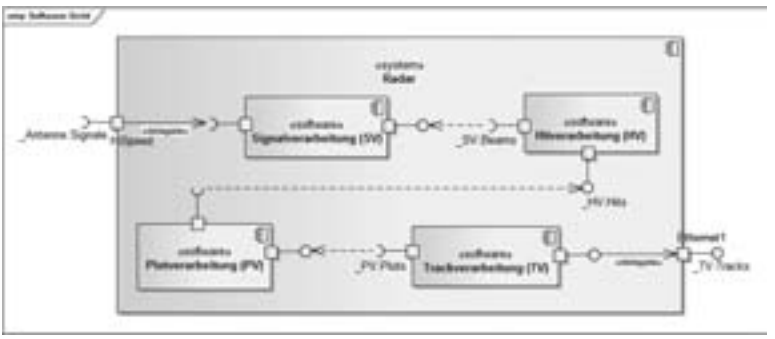


Abbildung 1: Software-Sicht des Radar-Systems im Entwicklungsmodell

Bei der Radarverarbeitung müssen große Mengen an Daten kontinuierlich und in Echtzeit verarbeitet werden. Die früher fast nur aus Hardware bestehenden Systeme werden heutzutage meist durch eingebettete Systeme realisiert. In der System-Architektur muss festgelegt werden auf welcher Verarbeitungseinheit die Software-Komponenten ausgeführt werden, welche logischen Schnittstellen diese haben und über welche physikalischen Schnittstellen diese übertragen werden. Ein im Entwicklungsmodell dokumentierter Entwurf des Architekten ist in Abbildung 2 zu sehen.

Die zwei folgenden Anforderungen an das Radar-System werden in Kapitel 4 zur Verdeutlichung des Validierungsprozesses und den dazugehörigen Begriffe verwendet:

- Das System muss eingespeiste Testsignale innerhalb von 220ms verarbeiten. (A1)
- Die durchschnittliche Prozessorauslastung muss kleiner gleich 60% sein. (A2)

Das Beispiel dient der Verdeutlichung des Ansatzes, weshalb wir viele einflussnehmende Aspekte, z.B. Latenzzeiten durch Kommunikation, sowie die Beeinflussung der Signale durch die kontinuierliche Verarbeitung an dieser Stelle unberücksichtigt lassen.

4. Der Validierungsprozess

Das Hauptziel beim Entwurf der System-Architektur ist, den Anforderungen des Kunden gerecht zu werden. Dies kann durch die Validierung der Architektur gegenüber den architekturrelevanten Anforderungen sichergestellt werden. Abbildung 3 stellt in einem Aktivitätsdiagramm die dafür benötigten Schritte unseres Validierungsprozesses dar.

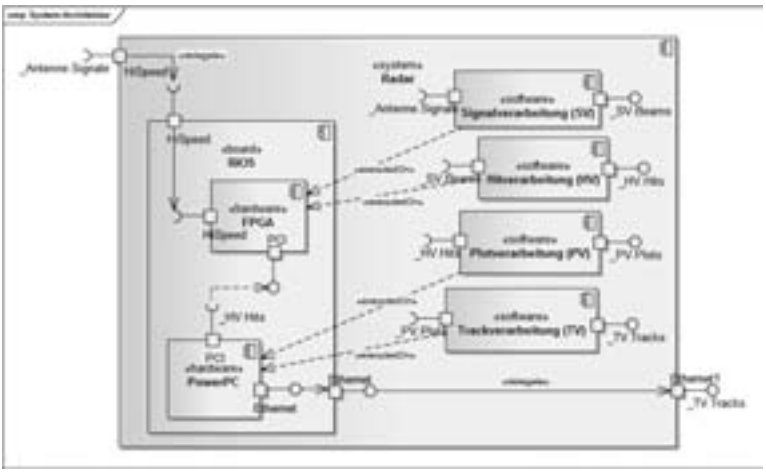


Abbildung 2: Zuordnung der Software-Komponenten zu den Verarbeitungseinheiten des Radar-Systems inklusive der physikalischen und logischen Schnittstellen

Aus der Menge der architekturrelevanten Anforderungen ermittelt der Architekt zunächst die Validierungsziele. Ein Validierungsziel beschreibt einen architekturspezifischen Aspekt auf einem für die System-Architektur passenden Abstraktionslevel. Dem Validierungsziel ist mindestens eine Anforderung zugeordnet. Zu jedem Validierungsziel gehört ein vom Architekten festgelegtes Validierungsverfahren, mit dem das Validierungsziel gegenüber den zugeordneten Anforderungen getestet werden kann. Im vorgestellten Radar-Beispiel aus Kapitel 3 ermittelt der Architekt aus den beschriebenen Beispielanforderungen A1 und A2 die Validierungsziele

- „Gesamtverarbeitungszeit“ (VZ1) mit der zugeordneten Anforderung A1 und
- „Durchschnittliche Prozessorauslastung“ (VZ2) mit der zugeordneten Anforderung A2.

Im nächsten Schritt erarbeitet der Architekt ein Validierungsmodell. Für jedes Validierungsziel legt der Architekt ein Validierungsverfahren fest und erstellt eine validierungszielspezifische Notation, in der er definiert, wie die für das Validierungsverfahren benötigten Informationen im Validierungsmodell zu modellieren sind. Die Eingabe der Daten kann durch die Übernahme vorhandener Daten aus dem Entwicklungsmodell und durch eine manuelle Eingabe erfolgen. Das fertige Validierungsmodell enthält also die für die Durchführung aller Validierungsverfahren benötigten Informationen auf Basis der jeweiligen validierungszielspezifischen Notationen.

Die Erstellung des Validierungsmodells soll zunächst am Radar-Beispiel verdeutlicht werden. Der Architekt muss jeweils ein Validierungsverfahren und eine validierungszielspezifischen Notation für die Validierungsziele VZ1 und VZ2 festlegen und die dazugehörigen Informationen im Validierungsmodell modellieren. Zur Validierung von VZ1 muss die Gesamtverarbeitungszeit des Systems ermittelt werden, die sich in diesem Beispiel aus der Summe der Verarbeitungszeiten aller Software-Komponenten i ergibt (siehe Formel 1).

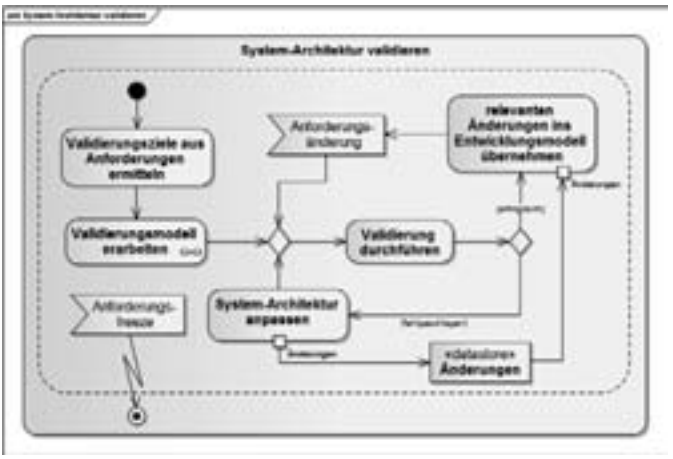


Abbildung 3: Ablauf der Validierung der System-Architektur

Formel 1: Gesamtverarbeitungszeit

$$executionTime_{total} = \sum_{i=1}^{n} executionTime_i$$

Formel 2: durchschnittliche Prozessorauslastung

$$processorLoad_p = \sum_{i=1}^{n} \frac{executionTime_i}{n}$$

Die Validierung ist erfolgreich, falls der Wert kleiner gleich des in A1 festgelegten Wertes (220ms) ist, ansonsten schlägt sie fehl. Es wird angenommen, dass dem Architekten die Verarbeitungszeiten der Software-Komponenten auf den verwendeten Verarbeitungsbausteinen z.B. aus früheren Projekten bekannt sind. Für dieses Verfahren werden die Software-Komponenten, die Verarbeitungsbausteine, die Zuordnung der Software-Komponenten zu den Verarbeitungsbausteinen sowie der Verarbeitungszeitwert der einzelnen Software-Komponenten benötigt. Bis auf die Verarbeitungszeitwerte sind alle Informationen bereits im Entwicklungsmodell vorhanden (siehe Abbildung 2), so dass der Architekt die verwendete Notation der benötigten UML-Elemente in die validierungsspezifische Notation übernimmt. Vervollständigt wird die validierungszielspezifische Notation von VZ1 durch ein UML-Profil (siehe Abbildung 4), welches den Software-Komponenten einen Tagged-Value *executionTime* zur Modellierung der Verarbeitungszeitwerte hinzufügt. Auf Basis dieser Notation erstellt der Architekt im Validierungsmodell das in Abbildung 5 dargestellte Komponentendiagramm.

Für das Validierungsziel VZ2 muss die durchschnittliche Prozessorauslastung einer jeden Verarbeitungseinheit p ermittelt werden. Sie ergibt sich vereinfacht aus der Summe der Verarbeitungszeiten der Software-Komponenten i geteilt durch die Anzahl aller auf p verarbeiteten Software-Komponenten n (siehe Formel 2). Die Validierung ist erfolgreich, falls der Wert kleiner gleich des in A2 festgelegten Wertes (60%) ist, ansonsten schlägt sie fehl. Für VZ2 kann die validierungsspezifischen Notationen von VZ1 wiederverwendet werden. Durch sie sind bereits alle benötigten Informationen im Validierungsmodell. Nach der Erarbeitung des Validierungsmodells durch den Architekten folgt im nächsten Schritt die Validierung der System-Architektur (siehe Abbildung 3). Diese setzt sich aus der Validierung aller Validierungsziele zusammen.

Schlägt eine der Validierungen fehl, muss der Architekt die System-Architektur anpassen und die Validierung erneut durchführen. Die für das Entwicklungsmodell relevanten Änderungen an der System-Architektur, z.B. Änderung der Zuordnung einer Software-Komponente zu einer Verarbeitungseinheit, werden dabei zwischengespeichert und im Falle einer erfolgreichen Validierung in das Entwicklungsmodell übernommen.

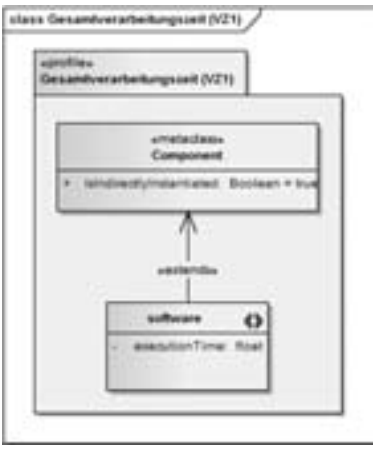


Abbildung 4: UML-Profil für VZ1 (Die Darstellung ergibt sich durch das verwendete Werkzeug Enterprise Architect [Spa10] und ist nicht UML-konform)

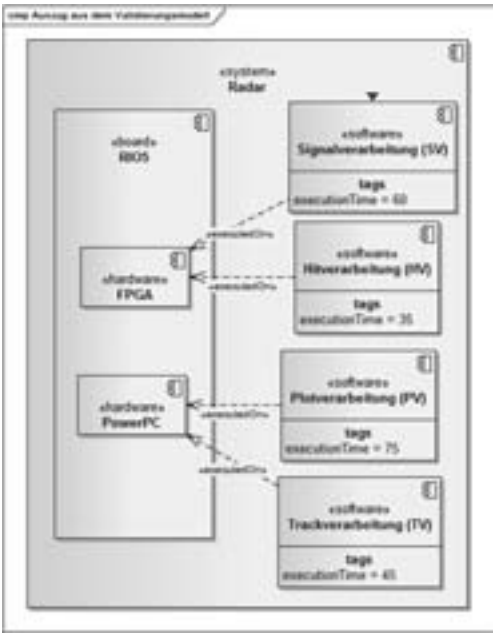


Abbildung 5: Auszug aus dem Validierungsmodell

Zum Abschluss dieses Kapitels werden nun die Validierungsverfahren von VZ1 und VZ2 mit konkreten Verarbeitungszeiten für die vier Software-Komponenten des Radar-Systems durchgeführt und danach die Auswirkungen einer Anforderungsänderung gezeigt. Für die Verarbeitungswerte werden die in Abbildung 5 modellierten Werte verwendet. Die Ergebnisse der Validierung beider Validierungsziele sind in Tabelle 1 und 2 zu sehen. Die Validierung der System-Architektur ist erfolgreich, da die Gesamtverarbeitungsdauer von 215ms kleiner als die in A1 festgelegten 220ms ist und die durchschnittliche Prozessorauslastung des FPGA (44%) und des PowerPCs (56%) je kleiner als die in A2 festgelegten 60% ist. Eine Veränderung der maximalen Gesamtverarbeitungsdauer für die Testsignale in Anforderung A1 auf 170ms hat folgende Auswirkungen: Die Veränderung des Vergleichswertes bewirkt keine Veränderungen an den Validierungszielen und den dazugehörigen Validierungsverfahren. Nach dem in Abbildung 3 beschriebenen Validierungsprozesses erfolgt nach einer Anforderungsänderung die erneute Validierung der System-Architektur, in unserem Beispiel also der Validierungsziele VZ1 und VZ2. Die Validierung der System-Architektur schlägt fehl, da die Validierung von VZ1 fehlschlägt, 215ms ist größer als 170ms. Der Architekt ordnet daraufhin die Software-Komponenten „Plotverarbeitung“ dem FPGA zu und führt die Validierung erneut aus. Die Ergebnisse sind in Tabelle 3 und 4 zu sehen.

Tabelle 1: Ergebnis des Validierungsverfahrens von VZ1

Software-Komponente	Verarbeitungseinheit	Verarbeitungszeit [ms]
Signalverarbeitung	FPGA	60
Hitverarbeitung	FPGA	35
Plotverarbeitung	PowerPC	75
Trackverarbeitung	PowerPC	45
Gesamtverarbeitungszeit [ms]		**215**

Tabelle 2: Ergebnis des Validierungsverfahrens von VZ2

Verarbeitungseinheit	durchschnittliche Prozessorauslastung [%]
FPGA	44
PowerPC	56

Durch die Änderung konnte zwar VZ1 erfolgreich validiert werden, 165ms ist kleiner gleich 170ms, jedoch hat die Änderung der Zuordnung Auswirkungen auf VZ2. Die Validierung der System-Architektur schlägt erneut fehl. Die Validierung schlägt fehl, da 73% größer den in A2 geforderten 60% ist. Das Problem kann beispielsweise durch die Aufteilung der „Plotverarbeitung“ in zwei Software-Komponenten gelöst werden. Das Beispiel zeigt, dass durch die erneute Validierung aller Validierungsziele auch die nicht direkt für den Architekten ersichtlichen Auswirkungen seiner Änderung erkannt werden.

Tabelle 3: Ergebnis des Validierungsverfahrens von VZ1 nach der Anforderungs- und System-Architektur-Änderung

Software-Komponente	Verarbeitungseinheit	Verarbeitungszeit [ms]
Signalverarbeitung	FPGA	60
Hitverarbeitung	FPGA	35
Plotverarbeitung	FPGA	25
Trackverarbeitung	PowerPC	45
Gesamtverarbeitungszeit [ms]		**165**

Tabelle 4: Ergebnis des Validierungsverfahrens von VZ2 nach der Anforderungs- und System-Architektur-Änderung

Verarbeitungseinheit	durchschnittliche Prozessorauslastung [%]
FPGA	73
PowerPC	27

5. Vorteile und Erfahrungen

Die Vorteile des vorgestellten Ansatzes können wie folgt skizziert werden:

- Die Automatisierung des Ablaufs von Validierungsverfahren reduziert den Arbeits- und Zeitaufwand für jede Validierung.
- Der Architekt kann die Auswirkungen von Anforderungsänderungen (Impact-Analyse) durch deren Verbindung zu Validierungszielen bis zur System-Architektur nachverfolgen.
- Der Validierungsprozess zeigt dem Architekten nicht direkt ersichtliche Auswirkungen, die durch Änderungen an der System-Architektur entstehen.
- Das Abstraktionsniveau der Validierungsziele ist beim Entwurf und bei der Analyse von System-Architektur-Änderungen hilfreich, damit der Architekt nicht den Überblick durch die hohe Anzahl an Anforderungen und deren Verbindungen untereinander verliert.

- Die Vergleichswerte für das Validierungsverfahren eines Validierungsziels stehen in den zugeordneten Anforderungen. Das Validierungsziel und das dazugehörige Verfahren bleiben bei Änderungen an diesen unverändert (siehe Anforderungsänderung am Ende von Kapitel 4).
- Die standard-konforme Verwendung der UML erlaubt die Transformation und das Auslesen von Modellinformationen mit vorhandenen Technologien wie beispielsweise dem Eclipse Modeling Framework [EMF10], was den entstehenden Mehraufwand durch den Ansatz reduziert und die Nutzung des UML-Modells als Datenquelle für die Validierungsverfahren mit geringem Aufwand ermöglicht.
- Die Auftrennung in Entwicklungs- und Validierungsmodell erlaubt eine unabhängige Bearbeitung der Daten und verhindert die Vermischung von validierungs- und entwicklungsspezifischen Informationen.
- Der Validierungsprozess kann bereits zu Beginn oder vor dem Projektstart den Architekten bei der Abschätzung der Entwicklungszeit und -kosten unterstützen. Die Validierung verwendet die aktuell in der Entwicklung vorhandenen Anforderungen und Informationen und kann dem Fortschritt im Projekt angepasst werden.

Der Ansatz wurde anhand von Projekten der SOPHIST GmbH und mit Hilfe unserer Erfahrungen aus verschiedenen Entwicklungen eingebetteter und softwareintensiver Systeme entwickelt. In einem der Projekte werden mehr als 40 Verarbeitungsbausteine, Mikroprozessoren und Field Programmable Gate Arrays (FPGA), und über 60 Software-Komponenten eingesetzt. Die dafür eingesetzten Validierungsverfahren berücksichtigen deutlich mehr Einflussfaktoren (z.B. Datenmengen, Datenreduktion, Kommunikations-Overhead und -Latenzzeiten, Speicherzugriffe, Seiteneffekte durch kontinuierliche Signalverarbeitung) als in dem gezeigten Beispiel und auch das Validierungsmodell besteht nicht nur aus wenigen Elementen, die sich in einem Diagramm darstellen lassen. Unsere Erfahrungen zeigen, dass sich der durch den Ansatz verursachte Mehraufwand bereits nach wenigen Iterationen des Validierungsprozesses amortisiert.

6. Zusammenfassung und Ausblick

Die Entwicklung eingebetteter softwareintensiver Systeme ist mit sich ständig ändernden Bedingungen konfrontiert, was gerade den Architekten der System-Architektur, dem Fundament der eingebetteten Systeme, vor große Herausforderungen stellt. Zu seinen Aufgaben zählen unter anderem der Entwurf und die Dokumentation einer System-Architektur unter Berücksichtigung vieler Einflussfaktoren sowie die Validierung dieser gegenüber den Kundenanforderungen. Diese zeit- und arbeitsaufwendige Aufgabe sollte gerade bei der zunehmenden Komplexität der zu entwickelnden Systeme, hohe Anzahl an Anforderungen und starker Verbindung dieser untereinander, nicht auf der Methode des scharfen Hinsehens basieren.

Der in diesem Paper vorgestellte Ansatz beschreibt einen Validierungsprozess, der in das iterative Vorgehen beim Entwurf der System-Architektur eingebaut wird. Mit Hilfe von Validierungszielen werden architekturrelevante Anforderungen auf einem für den Entwurf der System-Architektur passenden Abstraktionsniveau beschrieben.

Die Validierung der System-Architektur wird aufgeteilt in die Validierung aller Validierungsziele. Jedes Validierungsziel wird durch ein vom Architekten festgelegtes Validierungsverfahren gegenüber den Anforderungen getestet. Die für die Validierungsverfahren benötigten Informationen werden auf Basis einer vom Architekten festzulegenden validierungszielspezifischen UML-Notation im Validierungsmodell, unabhängig von den entwicklungsspezifischen Daten im Entwicklungsmodell, dokumentiert. Die entstehende Dokumentation ermöglicht die Nachvollziehbarkeit von Änderungen an der System-Architektur. Der Ansatz reduziert den Zeit- und Arbeitsaufwand für die Validierung der System-Architektur, indem der Ablauf der Validierungsverfahren automatisiert wird. Als Datenquelle dient das Validierungsmodell, deren Informationen durch die standardkonforme Verwendung der UML mit geringem Aufwand ausgelesen werden können. Es lassen sich dadurch auch Daten zwischen Entwicklungs- und Validierungsmodell austauschen. Durch die Entkopplung der Validierungsziele von konkreten Werten in den Anforderungen, müssen diese wie bei Wertänderungen nicht angepasst werden. Der durch den Ansatz entstehende Mehraufwand amortisiert sich nach unseren Erfahrungen bereits nach wenigen Validierungen, durch die Unterstützung des Architekten beim Entwurf und bei der Analyse von Anforderungs- und Architekturänderungen.

Aktuell wird an der Implementierung eines grafischen Werkzeugs für die Verwaltung der Validierungsziele sowie deren Verbindungen zu den architekturrelevanten Anforderungen und den Validierungsverfahren gearbeitet. Die oft noch manuell ausgeführten Validierungsverfahren sollen automatisiert werden. Ziel ist die Verbesserung der Benutzerfreundlichkeit des Ansatzes. In der Zukunft werden wir uns auf Simulationen zur Validierung verschiedener, stark voneinander abhängiger Validierungsziele konzentrieren. Diese Simulationen sollen auf mehr oder weniger detaillierte Informationen in Abhängigkeit von der aktuellen Entwicklungsphase basieren und somit dem Fortschritt im Projekt angepasst werden können. Dabei werden auch die Möglichkeiten unterschiedlicher Datenquellen, innerhalb und außerhalb der UML sowie deren Einbindung in die modellgetriebene Validierung der System-Architektur untersucht.

Literaturverzeichnis

[Aqu11] PREEvision von aquintos, http://www.aquintos.com, letzter Zugriff 04.02.2011

[Bal02] Marc Balcer, Stephen Mellor, Executable UML: A foundation for Model-Driven Architecture, Addison-Wesley, 2002

[CMU11] Architecture Analysis & Design Language (AADL), Carnegie Mellon University, http://www.aadl.info, letzter Zugriff 03.01.2011

[CoF10] CoFluent Studio, http://www.cofluentdesign.com/, letzter Zugriff am 28.08.2010

[EMF10] Eclipse Modeling Framework (EMF), Eclipse Foundation, http://www.eclipse.org/emf/, letzter Zugriff am 28.08.2010

[Fei10] P. Feiler, L. Wrage, J. Hansson, System Architecture Virtual Integration: A Case Study, Embedded Real-time Software and Systems Conference, Mai 2010

[OMG10a] System Modeling Language (SysML), Object Management Group, http://www.omgsysml.org/, letzter Zugriff am 28.11.2010

[OMG10b] UML-Profil MARTE, http://www.omgwiki.org/marte/, Object Management Group, letzter Zugriff am 28.11.2010

[Spa10] Enterprise Architect, Sparx Systems, http://www.sparxsystems.com, letzter Zugriff am 28.11.2010

[UML10] Unified Modeling Language (UML) Version 2.3, http://www.uml.org, 2010



Evolutionäre Software- und Systementwicklung – Methoden und Erfahrungen (ESoSyM-2011)

Evolutionäre Software- und Systementwicklung - Methoden und Erfahrungen (ESoSyM-2011)

Workshop der Konferenz SE2011 und Fachgruppentreffen der Fachgruppe OOSE der GI

Ziele: Bei der Entwicklung von Softwaresystemen sind immer komplexere Anforderungen zu erfüllen. Gleichzeitig steigen die Forderungen nach langfristiger Nutzbarkeit bei einfacher Änderbarkeit der Systeme. Evolutionäre Entwicklung und modellbasierte Methoden helfen, diese Ziele zu erreichen und Risiken der Entwicklung besser zu beherrschen. Allerdings bestehen Differenzen zwischen der Entwicklung von Ansätzen in der Forschung und der Anwendung in industrieller Softwareentwicklung. Der Workshop soll breiten Raum für die Erörterung problemangepaßter Ansätze und Ergebnisse der Forschung sowie der Problemstellungen und Erfahrungen der Industrie bieten. Darüber hinaus soll er auch eine Zusammenfassung und Darstellung der Arbeiten der Arbeitskreise der Fachgruppe OOSE ermöglichen.

Schwerpunkte:

* Modellbasierte Methoden für evolutionäre Entwicklung
* Methoden und Ansätze zur Unterstützung langlebiger Systeme
* Methoden mit Unterstützung für spezifische Eigenschaften eingebetteter Systeme
* Produktlinien- und Komponenten-Methoden
* Effizienz der Entwicklungstätigkeiten im Software-Lebenszyklus
* Verbindung zwischen Werkzeugen
* Unterstützung bei Entwurfsentscheidungen
* Transfer und Anpassung von Forschungsergebnissen an betriebliche Erfordernisse
* Erfahrungsberichte aus Einführung und Anwendung von neuen Ansätzen

Organisatoren:

Matthias Riebisch, TU Ilmenau
Gregor Engels, Universität Paderborn
Bernhard Schätz, fortiss gGmbH, München
Christian Zeidler, ABB Forschungszentrum Ladenburg

Programmkommitee:

Gregor Engels, Universität Paderborn
Wilhelm Hasselbring, Universität Kiel
Andrea Hermann, Axivion GmbH
Carola Lilienthal, C1 WPS Workplace Solutions GmbH, Hamburg
Barbara Paech, Universität Heidelberg
Ralf Reussner, KIT Karlsruhe
Matthias Riebisch, TU Ilmenau
Bernhard Schätz, fortiss gGmbH, München
Detlef Streitferdt, TU Ilmenau
Peter Tabeling, InterVista AG
Christian Zeidler, ABB Forschungszentrum Ladenburg
Heinz Züllighoven, Universität Hamburg

On the Impact of Goals on Long-Living Systems

Arfan Mansoor, Detlef Streitferdt

{arfan.mansoor | detlef.streitferdt}@tu-ilmenau.de

Abstract: Changes in software systems are inevitable. Addressing future changes in the requirements engineering phase influences the resulting architecture, improves its maintainability and can prolong the lifetime of software architectures. In this paper, structures in goal models are identified and discussed. Such structures are parts in requirements models with a high probability for future changes. Designing a system to support changes is a way towards a longer lifetime. Variable requirements model parts can be handled using adjustable parts in the design. Thus, the better current as well as future variation points can be identified in the requirements model, the better changes can be handled by a system and, consequently, the longer the lifecycle of software architectures could be.

1 Introduction

The first important step towards developing a system is requirement engineering. Changing requirements while the system is developed, as well as changes over the lifetime of a system, are part of the daily work in software development [Som00]. Thus, the better changes are addressed or foreseen in the requirements engineering phase of the initial system, the longer the lifetime of a system could be, since the system is able to address many changes "by design". Moving from requirements to the system design, a proactive approach to address changes are product lines [BKPS04]. Here, from the beginning on the system is designed for changes, which are used to derive applications based on the product line.

The lifetime of a software system is mainly influenced by the changes that happen over time, especially changes not within the standard use of the software. Changes of requirements or changes of the hardware platform will most likely disrupt the architecture of the software. To prolong the time before an architecture is not usable any more, as many variabilities as possible need to be identified in the requirements engineering phase. After a thorough analysis, such variabilities may be part of the design and thus, the probability to disrupt the architecture can be reduced.

In this paper, a goal-oriented requirements approach with a goal model is used to address possible changes in the requirements engineering phase. Structural patterns in the goal model are used, which refer to the variable parts of the system identified in an early development phase. With a relation of such a variable goal model to a feature model of a product line, a highly configurable system architecture can be developed.

The approach is based on an extensive analysis of the current and future requirements and goals of the system. Based on this analysis a roadmap containing the different versions of the system over time will be developed. All the different versions are based on the same software architecture, in our case a product line, which supports all versions from the beginning on. Thus, changes, starting in the requirements engineering phase, are built into the system. Addressing possible future requirements in the system design ensures their seamless integration into the system over the complete lifetime. The value of the goal model is its structure for variable goal sets, which can be directly transformed into features as described in this paper.

2 State of the Art

Requirements engineering approaches so far are in the what-how range. The idea of goals has introduced "why" concerns in the early stage of requirement engineering i.e., "why" the system was needed and does the requirement specification captures the needs of stakeholders. The idea of goals also emphasized the understanding of organizational context for a new system [Lap05]. Goal based requirements engineering is concerned with the identification of high level goals to be achieved by the system envisioned, the refinement of such goals, the operationalization of goals into services and constraints and the assignment of responsibilities for the resulting requirement to agents such as human, devices and programs [DvL96]. Moving from goal models to the system design, the goal model structures referring to variable elements are responsible for the future capability of a system to react upon changes. The combination of goal models and product lines is very promising to continuously address variability within system development.

Highly configurable, mass customizable systems, are often based on product lines having a core and several, user selectable variabilities. The core and the variabilities are modeled with features, first described 1990 by Kyo C. Kang and further developed in [KKL+98] and [CE00]. Features describe the product line for a future user, so that the user can choose an own application, based on the features of the family. They should describe an outstanding, distinguishable, user visible aspect, quality or characteristic of a software system or system. Features are arranged as a tree, the root of the tree is the concept node, representing the product line itself. Every feature can be optional or mandatory. By definition, product lines are used for many systems and our assumption is, they have a longer lifetime than single systems. Of course, the best support for a long lifetime is to prepare the product line to handle as many of the future changes as possible. Thus, as more degrees of freedom are built into the product line, more changes can be addressed. Sure enough, the degrees of freedom must be based on a thorough requirements analysis phase.

Using goal models to derive feature models is described in [UHS09]. There, the goal models of several system variants are merged and analyzed for common and variable goals. This commonality/variability analysis leads to common (mandatory) and variable (optional) features. The structure of the goal model is kept stable. In addition to the approach described above, the analysis of goal structures together with the clustering of

goals, as described in this paper, leads to an enhanced feature model, with the support for additional future changes. In this paper, the combination of goal oriented requirements engineering and feature modeling results in an enhanced version of the feature model for a product line, addressing the possible variations of the initial goal model. With the analysis and extension of the variable parts in the goal model we expect the resulting product line to have a longer lifetime compared to a product line without the integration of future changes.

3 Variabilities in Goal Models related to Feature Models

The example in this paper is a cycle computer system. This system will be attached to a bicycle, will process data from various sensors, and will communicate with a standard PC. A cyclist will be supported while riding the bike, for maintenance issues, for tour preparations, or to enhance the safety using the bike. One of the results of the requirements engineering phase is a goal model.

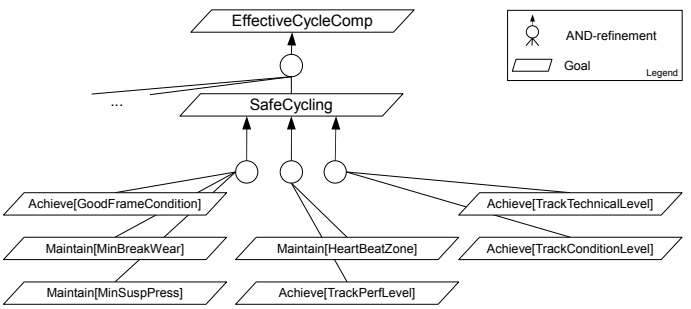


Figure 1: Cycle Computer Goal Model (partly)

As drawn in Figure 1 and part of the overall goal model of the cycle computer, the safety goal has three alternative clusters of subgoals, which are based on the idea of three ways to achieve safety while riding the bicycle. The selected goals per cluster are the result of a goal workshop and may be subject to change in the future development.

The first alternative cluster relates to the technical safety of the bike. To achieve a good frame condition (Achieve[GoodFrameCondition]), any torsion of the frame is permanently measured and analyzed to calculate the current wear level. Thus, the probability of a cracked frame can be derived and used to inform the cyclist of upcoming hazards. Especially in the downhill domain, it is important to know whether the frame would most probably survive the next race. For an enhanced safety information, the wear levels of the brake pads need to be maintained at a minimum level (Maintain[MinBreakWear]) to ensure the proper function. Finally, the suspension pressure also needs to be maintained above a minimum level (Maintain[MinSuspPress]) to prevent serious damage on the bike and possible accidents.

The second alternative of safety is related to cyclists with cardiovascular diseases. Two goals need to be satisfied for a safe bicycle tour. First, the cyclist will be informed whether his current heartbeat rate is inside the heartbeat zone (Maintain[HeartBeatZone]) discussed with the doctor. Thus, this zone needs to be maintained throughout the complete cycle tour, to ensure a healthy trip for the cyclist. In addition, for any planned trip, its performance level (Achieve[TrackPerfLevel]) needs to fit to the cyclists capabilities. Such a selection of trips ensures to stay inside individual heartbeat zones.

The third group of goals refers to the safety of a chosen track. For each part of the track the technical level (Achieve[TrackTechnicalLevel]) should be known, or in other words, what driving experience is needed to handle the track safely. For the complete track, the technical level should fit to the cyclists capabilities. Finally, the condition of the track (Achieve[TrackConditionLevel]) should fit to the cyclists fitness level. Single trails, gravel paths, or paved roads result in different performance requirements on the cyclist, even though the length and gradients of a trip may be the same.

The identification of possible changes at the requirements level is important to address changes properly in the design phase. In the goal model, the relation of goals to subgoals, with subgoals being arranged in clusters, is a very promising pattern for future changes. This structural pattern may be present as alternatives, shown in Figure 1, or as standard goal-subgoal relation. For the cycle computer example the identification of the clusters was based on the domain knowledge and on the different types of cyclists identified in the use case analysis.

With the goal model described above, a first level of variability is achieved by the three alternative clusters of the goal model. Adding new alternatives to satisfy the safety goal is an easy extension of the goal model. The structure of the goal model will not suffer. Adding new goals to a given alternative would also keep the structure stable. The relation of the goal model to a feature model can answer the question for the extensibility of the design. Feature levels in feature models can be used to identify architectural layers and feature subtrees can refer to components. Such components are parameterized (with different binding times) to meet different requirements and realize the derivation of products out of the product line.

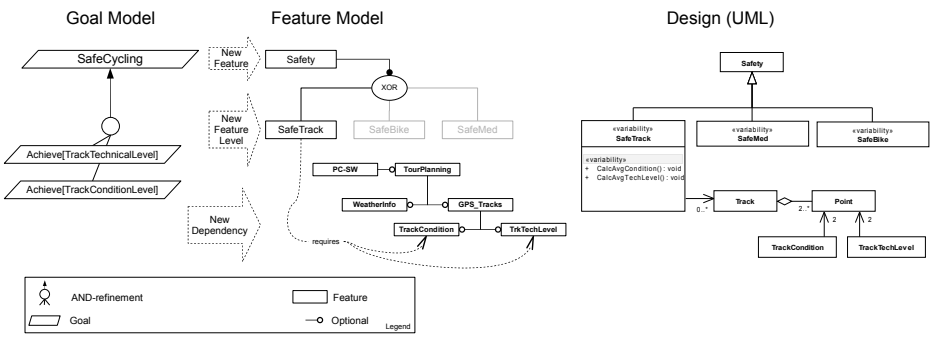


Figure 2: Relation of Goals to Features

For the cycle computer, the proposal for the relation of goals and goal structures is shown in Figure 2. The high level goal of reaching safe cycling trips results directly in a newly introduced feature "Safety". This feature is the root feature of a new feature subtree resulting in a component of the product line, the internal design of the component is shown on the right side of Figure 2. On the next level, each alternative in the goal model is represented by a corresponding feature. Here, three new features, "SafeTrack", "SafeBike", and "SafeMed" are organized below the safety feature. The alternative clusters of the goal model are resolved as XOR-relation in the feature model, to conform to the initial goal model. In case the alternatives are not mutually exclusive, the feature "Safety" would have three optional subfeatures. The subgoals on the left side of Figure 2 are resolved by two new "requires"-dependencies. They refer to the corresponding features in the feature tree of the cycle computer to enable cycling tracks with the needed information about the technical level and the track condition. Finally, as drawn on the right side of Figure 2, the feature model is realized by the UML class model. Here, the realization of the variabilities in the feature model is achieved by generalizing the "Safety"-class with the concrete safety feature. The stereotype <<variability>> together with the generalization is used to realize the XOR-relation of the feature model. Based on the "requires"-relationship, the features "TrackCondition" and "TrkTechLevel" are selected and present in the UML class model.

The identification of variable structures in the goal model points to model parts, where changes can be made without causing structural damage to the overall model. Such parts need to get more emphasis by additional future workshops to identify as many future goals as possible. The result is an enhanced goal model, which can be used as roadmap of the intended system, reaching in the far future. Long living architectural structures should be designed for changes. Feature models can handle the core and any variable parts of a product line. A component-oriented approach to design a product line is common. Thus, changes on the component level of product lines are expected. Relating goal models to feature models leads to additional components and addresses variability inside the component as another level for possible future changes. Thus, a system based on a goal oriented requirements approach and a partly derived feature model, as discussed above, can handle more changes. With this, we expect a longer lifetime for systems based on such an architecture. The benefit of the approach lies in the support of the product line design, namely the feature model. Based on the goal model, the feature model will be enhanced with new structures, which introduce additional levels of variability. As discussed, the level of variability, especially variability in relation to future goals, results in a design supporting the roadmap of the product with its future changes without breaking the software architecture. Automated transformations from the goal model to the feature model are the subject of current research efforts. They will reduce the development time.

4 Summary and Outlook

In this position paper an approach was presented to use the goal model of goal-oriented requirements engineering and identify structural parts in the goal model, which have a high

probability for changes (goals are either added, changed, or removed). These structures are goal-subgoal relations with subgoals that can be clustered in logical groups, all satisfying the upper goal with a different strategy. These structures have been mapped to a feature model, by introducing new features and "requires"-relations. The new features refer to a new system component parameterizable according to the initial goals. The thorough analysis of future requirements and goals will add the variability for future changes to the goal model. The feature model is enhanced to support the future variabilities without breaking the software architecture. Finally, automated transformations from goal models to feature models, which are subject of current research efforts, can reduce the development time. This approach is promising for long living systems, since the resulting product line architecture will address more changeable goals of the initial requirements engineering phase. The formalization of the approach, its full integration into goal oriented requirements engineering and the validation by additional examples are future research topics.

References

[BKPS04] Günter Böckle, Peter Knauber, Klaus Pohl, and Klaus Schmid. *Software Produktlinien*. dpunkt.verlag, 2004.

[CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker, editors. *Generative and Component-Based Software Engineering, First International Symposium, GCSE'99, Erfurt, Germany, September 28-30, 1999, Revised Papers*, volume 1799 of *Lecture Notes in Computer Science*. Springer, 2000.

[DvL96] Robert Darimont and Axel van Lamsweerde. Formal Refinement Patterns for Goal-Driven Requirements Elaboration. In *SIGSOFT FSE*, pages 179–190, 1996.

[KKL+98] Kyo Chul Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. *Ann. Software Eng.*, 5:143–168, 1998.

[Lap05] A. Lapouchnian. Goal-Oriented Requirement Engineering. An Overview of the Current Research. *Technical Report, University of Toronto*, 2005.

[Som00] Ian Sommerville. *Software Engineering 6th Edition*. Addison Wesley, 2000.

[UHS09] Kohei Uno, Shinpei Hayashi, and Motoshi Saeki. Constructing Feature Models Using Goal-Oriented Analysis. In Byoungju Choi, editor, *QSIC*, pages 412–417. IEEE Computer Society, 2009.

A Qualitative Discussion of Different Approaches for Implementing Multi-Tenant SaaS Offerings

Christof Momm, Rouven Krebs

SAP Research
Vincenz-Priessnitz-Str. 1
76131 Karlsruhe, Germany
christof.momm@sap.com
rouven.krebs@sap.com

Abstract: The upcoming business model of providing software as a service (SaaS) not only creates new challenges for service providers but also for software engineers. To enable a cost-efficient service management, the hosted application should support multi-tenancy. For implementing multi-tenancy, several options are available, whereas all of them to a certain degree require a reengineering of the application at hand. Since for many applications this represents the next big evolution step, this paper is devoted to discussing different options for implementing multi-tenancy regarding the initial reengineering effort required for meeting the customer expectations versus cost-saving potential.

Introduction

Today, we observe a paradigm shift in the way software is delivered to customers. While it used to be the common practice to ship software packages to the customer and operate the deployed software on premise, nowadays the consumption of certain software as on demand services (i.e. Software-as-a Service, short SaaS) becomes more popular [DW07]. Customers expect from such a SaaS offering flexibility in case of changing requirements, a contractually guaranteed quality of service based on service level agreements (SLAs) [TYB08] and a demand-oriented billing. Major SLAs for SaaS offerings include responsiveness, unplanned downtime (availability), planned downtime and data safety (in particular backup) guarantees.

Acknowledgement: This work is supported by the German Federal Ministry of Education and Research under promotional reference 01jG09004 (ValueGrids) and received funding from the European Union's Seventh Framework Programme FP7/2007-2013 (4CaaSt) under grant agreement n° 258862.

For service providers, the key success factor is to offer scalable demand-oriented services (also referred to as elastic cloud services) [BYV08] with high SLA compliance, while achieving low total cost of ownership (TCO), in particular low incremental costs. The TCO for customization, provisioning and operation of software can be significantly reduced by introducing multi-tenancy, where "user requests from different organizations and companies (*tenants*) are served concurrently by one or more hosted application instances based on the shared hardware and software infrastructure" [GS+07].

For implementing multi-tenancy, several options are available, reaching from shared single parts of the application stack, such as infrastructure or middleware, to shared application instances. The incremental operating costs for a tenant thereby highly depend on the level where multi-tenancy is introduced in the application stack. This is because each layer involves a common, non-tenant-specific overhead, which can be saved by introducing resource sharing. Thus, the shared application instances in the long term always represent the TCO-minimal solutions. However, at the same time each of these options to a certain degree requires the reengineering of existing applications, in particular if the resulting solution should meet customer expectations regarding quality of service and scalability. Hence, which option the software engineer / provider should decide on not only depends on the expected incremental operation costs, but also on the complexity of the necessary modifications for the given application in combination with the expected service lifetime as well as number of customers.

For many existing applications, multi-tenancy enablement (as part of a general SaaS enablement) represents the next big evolution step. Therefore, this paper aims at helping decision makers in comparing different multi-tenancy implementation options for a given SaaS scenario regarding the resulting TCO. To this end, we first point out major customer requirements as well as the most prominent options for multi-tenancy-enabling an existing application. Then, we introduce a simple cost model explaining the major cost drivers, both fixed and variable. Finally, we provide a qualitative discussion of the different implementation alternatives regarding the fixed initial reengineering effort (fixed costs) required for meeting customer expectations and the variable costs for operation. This discussion is based on our experiences with large-scale business applications and considers customer requirements such as (dynamic) scalability as well as guarantees regarding responsiveness, planned downtime and provisioning time. Availability, data safety, security requirements are not in scope.

Customer Requirements and Multi-Tenancy Implementation Options

This section introduces the considered customer requirements on SaaS offerings along with different feasible options for implementing a corresponding multi-tenant solution on the provider side.

Looking at existing SaaS offerings and talking to customers, we identified the following major requirements and multi-tenancy challenges:

- *Provisioning or System-ready-to-use time:* Defines the time for provisioning a new service instance to a customer. Customers thereby expect getting a fully-operational instance offered within minutes to hours, not hours to days.

- *Responsiveness:* For a given customer workload threshold, the application must not drop below a certain lower response time and/or throughput bound, e.g. sub-second average response times. In a multi-tenant setting different tenants share the same resources, which creates the risk that tenant workloads interfere with each other. An overload created by one tenant may therefore negatively impact the performance of another tenant. Accordingly, adequate performance isolation is required on the provider side.

- *(Dynamic) Scalability:* Customers expect from SaaS offerings that they adapt to changing workload requirements in a very flexible way. Ideally, such an adaptation happens fully dynamic (within seconds), but even a more static approach (within hours to days) in most cases is already sufficient.

- *Planned Downtime:* Since the software still evolves over time, the provider requires time for regular maintenance and upgrade activities. Customers expect guaranteed upper limits for downtimes and time frames, e.g. weekend only. Furthermore, it might happen that different customers expect different time frames, with only little or even without any overlap. A multi-tenant setting may introduce additional dependencies between tenant upgrade or patch activities leading to significantly higher time demands.

- *Availability (Unplanned Downtime):* Refers to the service uptime without planned downtimes. A customer-friendly definition of available ("up") would be that the client is able to use the processes of his scoping, which implies that all involved components are online. Due to the shared usage of resources a failure at a single point may cause the violation of several tenant SLAs at once.

- *Data safety:* Adequate mechanisms must be in place to avoid data loss. SLAs mostly include statements about backup mechanisms, in particular recovery point and recovery time objectives. Special backup mechanisms must be in place to support a tenant-specific rollback of data without overwriting non-affected data while ensuring data isolation.

- *Security/Privacy:* Data security and legal requirements regarding storing locations etc. must be ensured. In a multi-tenant setting the provider has to ensure a strict logic data isolation between tenants. Since tenants might share the same data base or even the same table the application logic's complexity might increase.

It is worth to mention, that the requirements themselves as well as their importance strongly depend on the customers size [SS+10].

In [KM08] and [OG+09] several options for implementing a cost-efficient multi-tenant solution are presented. In the following, we briefly introduce the option we think are generally suitable for meeting these requirements. Figure 1 illustrates the different options. All of these options have in common that still several instances of the shared part may exist, e.g. several physical servers with a VM installed or several shared middleware instances. Hence, the allocated elements may be moved to another instance of the shared resources, which represents one way for (dynamically) scaling of resources.

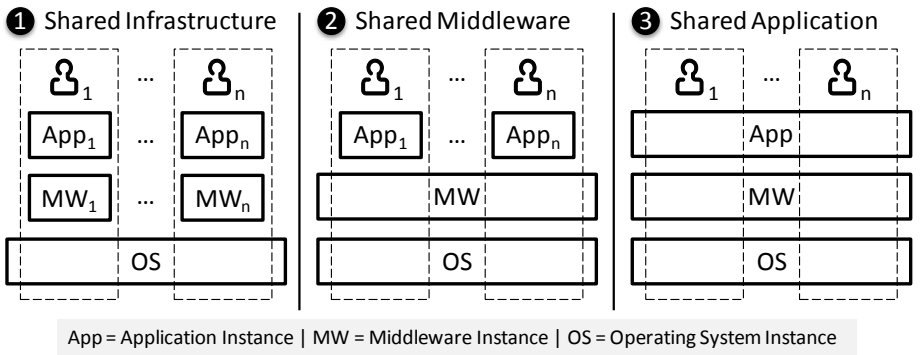


Figure 1: Considered Multi-Tenancy Implemenation Options

In the following, we briefly introduce the specifics of the different options.

- *(1) Shared Infrastructure:* In this case, only the infrastructure is shared amongst tenants using virtualization technologies. To enable a convenient provisioning of tenant instances, virtual appliances comprising application and middleware instances have to be created. So data isolation is per se given, while performance isolation has to be ensured by the VM technology. Scalability can either be achieved by increasing resources assigned to a VM (scale up) or by moving a VM to another shared hardware (scale out). All application lifecycle management (LM) procedures (upgrade/patch) may remain as they are.

- *(2) Shared Middleware:* This option involves a shared middleware instance on top of a shared OS, which may be either placed on a physical or virtualized hardware. Data isolation in this case is also per se given and the LM procedures on the application level do not have to be changed. Fast provisioning, performance isolation and scalability by contrast have to be natively supported by the middleware. Regarding the scalability, both a scale up of the shared middleware instance (e.g. using VM technology) and a scale out of tenants to another shared middleware instance is feasible.

- *(3) Shared Application:* This option involves least overhead per tenant since already the application instance shared amongst tenants. However, at the same this option probably involves most reengineering activities. Data and performance isolation have to be added on the application level. Similarly, scalability and LM procedures might require significant changes of the application.

Simple Cost Model for Evaluation

Depending on the application that should be offered as service and the option used for implementing multi-tenancy, different reengineering measures of varying complexity are necessary for fulfilling this requirement. The goal of decision makers (i.e. service providers in cooperation with the responsible software engineers) is to find the cost-minimal solution for the considered application. This section therefore introduces a coarse-grained cost model that helps comparing different alternatives.

This cost model includes two major components:

- *Initial reengineering costs:* Initial costs for reengineering activities required once for implementing the chosen multi-tenancy option for a given application.

- *Continuous operating costs:* Monthly costs for operation. This includes fixed costs per instance (application, middleware or hardware) as well as costs per tenant.

Accordingly, every reengineering activity that reducing the monthly operating costs will sooner or later be amortized. This breakeven point is calculated as follows

$$NumMonthToBreakEven = \frac{Intial\ Reeng.Costs}{\Delta Operating\ Costs}$$

If the service is offered forever, the variant with the lowest incremental cost is always the best. However, we rather assume that a service sooner or later gets replaced. Thus, the question is if this time span is sufficiently long to justify huge investments (i.e. service usage time > time to break even). To determine the breakeven point the operating costs have to be estimated. These operating costs comprise monthly costs for basic tenant-independent efforts produced by the operated instances of the application stack instance (application, middleware or operating system) as well as tenant-specific application efforts. Hence, the operating cost function depends on the selected implementation option and the number of expected number of tenants n:

$$OpCost_{sharedInfra} = CostOS_{Base} + n * (CostMW_{Base} + CostApp_{Base} + Cost_App_{Tenant})$$

$$OpCost_{sharedMW} = CostOS_{Base} + CostMW_{Base} + n * (CostApp_{Base} + Cost_App_{Tenant})$$

$$OpCost_{ShareApp} = CostOS_{Base} + CostMW_{Base} + CostApp_{Base} + n * Cost_App_{Tenant}$$

Accordingly, introducing an additional shared resource in the stack saves $\Delta Operating\ Costs$ of (n-1) times the base costs for the shared resource. Factors influencing these base costs are resources demanded by the produced base load, maintenance efforts as well as lifecycle management efforts.

Note that these cost functions in reality might not be linear. However, it is probably not feasible to determine the exact cost functions and they are still suitable for comparing different options.

The single components of this cost function have to be estimated for the different options for multi-tenancy-enabling the application at hand. More precisely, the effort for the different required modifications (reengineering activities) *as well as* the expected maintenance/operations costs has to be estimated for each alternative. Some feasible approaches for estimating these efforts are discussed in section "Related Work". To support this effort estimation, in the following section we point out typical modifications required for multi-tenancy-enabling existing applications and discuss selected aspects regarding the resulting operating costs for the different implementation options.

Discussion of Reengineering Effort per Implementation Options

In this section, we discuss the potential reengineering effort for each multi-tenancy implementation as well as some aspects regarding the operating costs, in particular the expected resource consumption. The discussion is based on our experiences with large-scale business information systems and considers the application and middleware layer. The reason for this is that the middleware might be a proprietary development, which might have to be enhanced as well. The scope of the discussion is limited to certain customer requirements pointed out in section 2. Availability (Unplanned Downtime), data safety and security/privacy are not further discussed in view of the reengineering measures.

Shared Infrastructure

Initial Reengineering Effort:

Provisioning or System-Ready-to-Use Time
It should be possible to create new instances of the stack from master images. To this end, it might be necessary to relax existing copyright protection mechanisms, in particular hardware keys. Moreover, it must be possible to change addressing information dynamically after installation. Static addressing information has to be removed.
Responsiveness
Responsiveness in this case depends on amount of virtualized resources and performance isolation is handled by VM technology. It should be checked beforehand that the used VM technology does not cause additional performance problems. If so, it

might be necessary to change parts of the implementation, e.g. avoid certain memory operations.
(Dynamic) Scalability
To support a growing customer workload, the resources assigned to the respective VM can be increased up the limits of the physical hardware. Performing this VM-based scaling dynamically, i.e. without having to restart/reboot the system, has to be supported by the used middleware and operating system. Both have to detect additional resources (in particular CPU and memory) on the fly. Extending an existing middleware might be very complex. The application in turn has to be able to deal with multiple cores. Otherwise, this might cause significant reengineering effort.
Planned Downtime
Since each customer gets its own middleware and application instance, there are no side effects with other tenants. Thus, the upgrade procedures may remain the same as in an on premise setting as long as the upper limit regarding the planned downtime is sufficient. Otherwise, the corresponding LM procedures have to be revised as well.

Operating Costs (Resource Demand): "Shared Infrastructure" involves the lowest degree of resource sharing. Thus, providers per se have to deal with a significant overhead (VM, middleware and application base load). Furthermore, information about application internals is not available, which limits the potential of optimizing resource utilization and probably results in higher demands.

Shared Middleware

Provisioning or System-Ready-to-Use Time
The middleware has to support running multiple deployments of the same application components. Regarding the application level, just like in case of the shared infrastructure, it might be necessary to enable a dynamic addressing. Moreover, it is likely that configuration, monitoring and logging have to be revised in order to support multiple application instances on one middleware. Quite often, there is only one configuration/monitoring/logging file per application and multiple deployments of the same application instance are not supported.
Responsiveness
Concerning the responsiveness, the multiple application deployments may interfere with each other and by this create critical overload situation. To avoid this, the middleware has to provide adequate performance isolation mechanisms, e.g. fixed thread or memory quota per application instance (i.e. tenant) or tenant-aware request queuing. This should ideally be transparent to the application.
(Dynamic) Scalability
If no VM technology is used, one way for reacting on changing workload requirements is moving an application instance to another less occupied middleware instance. In case application caches exist, performance might drop significantly if these caches are not moved as well. A dynamic scaling would then require a live migration of application

instances including the complete application cache, just like it is already supported by some VM technology on the infrastructure level [CF+05]. Another option for managing workloads is to introduce a load balancing between application instances by actively managing their resource quotas.
Planned Downtime
In case of application upgrades which do not affect the middleware the upgrade procedures may remain the same as in an on premise setting as long as the upper limit regarding the planned downtime is sufficient. However, a middleware upgrade probably requires upgrading *all* application instances (tenants). To ensure that all of them finish within the given time limit, the existing upgrade procedures might have to be adapted, e.g. introduce parallel processing and avoid synchronization points.

Operating Costs (Resource Demand): In this scenario, hardware, operating system and middleware are shared and only the multiple application instance cause additional overhead. The resource optimization potential is better than in the "Shared Infrastructure" scenario, but still limited since application-specific information is missing. Moreover, the resource demand heavily depends on the available middleware capabilities. If live migration of application instances without cache loss is not supported, the demand is much higher.

Shared Application

Provisioning or System-Ready-to-Use Time
Tenants in this case are solely represented as a set of tenant-specific data, meta-data and configurations within an application instance. The application has to be modified in a way that new tenants may be created automatically, e.g. using pre-defined templates. As prerequisite, this tenant-specific information has to be isolated, which might not be trivial [OG+09]. More work on how to enable existing applications to support multi-tenancy is presented in section "Related Work".
Responsiveness
Tenant workloads necessarily interfere with each other when sharing a single application instance. Performance-isolation is strongly required, but implementing it solely on the application level is probably not feasible. Instead, the middleware has to expose an API for effectively assigning resources to tenant-specific transactions, whereas the application has to implement adequate mechanisms for controlling the tenant/resource allocation.
(Dynamic) Scalability
If a VM technology is used, it is possible to scale up the resources assigned to the application instance (given that changes in the infrastructure are recognized). In addition to this, tenants could be moved to another less busy application instance (scale out). Just like in case of the shared middleware, a tenant-move should then consider all tenant-specific cache data. The application has to be extended by adequate management interfaces, which allow easily retrieving this tenant data. For supporting a live migration

does probably not work without changing the middleware as well. At least the message dispatcher has to be aware of the two running application instances and perform the final switch.
Planned Downtime
For multi-tenancy-enabled applications, it is very likely that the LM procedures have to be adapted to distinguish between common and tenant-specific steps. In addition to this, developers have to consider the fact that all LM procedures have to be performed for the whole application instance in a transactional manner, but still involve tenant-specific activities. Ensuring adherence to the given planned downtime constraints might therefore require major revision to the existing LM procedures. Parallel processing of tenant upgrades and avoidance of synchronization points across tenants is crucial. Also, adequate roll-back mechanisms should be provided in case it is not possible to finish an upgrade/patch in time. Supporting tenant-specific timeframes in this case might not be feasible any more. The only option we can think of is putting together tenants with similar time frame requirements, meaning that upgrade constraints have to be considered within scope of the tenant placement.

Operating Costs (Resource Demand): This scenario involves the highest degree of resource sharing and therefore introduces the minimum possible overhead in terms of additional base load. Moreover, it offers the best resource optimization potential since very fine-grained application-level monitoring is possible. This particularly allows a better prediction of resource demands based on given customer workloads. However, again the optimization potential heavily depends on the maturity of the tenant lifecycle management operations, in particular the resource demand for tenant moves.

Related Work

To apply the simple cost model, adequate effort estimation techniques are required. In the following we point out some suitable approaches. The reengineering effort for evolution steps can for instance be predicted by using top-down effort estimation approaches such as *Function Point Analysis* (FPA) [IFP99] and evolution-specific extensions to the *Comprehensive Cost Model* (COCOMO) II [BD+03] or bottom-up effort estimation techniques like *Architecture-Centric Project Management* (ACPM) [PB01]. Predicting the maintenance and operation costs is probably more difficult. Besides a bottom-up estimation involving the operations team, a general evaluation of the application's maintainability might be helpful, such as the *Architecture-Level Prediction of Software Maintenance* (ALPSM) [BB99]. In addition to this, systematic measurements as for instance proposed in [WH10] could be used to determine the middleware / application base load.

A major component of the cost model we introduced represents the initial reengineering effort required for introducing multi-tenancy. Looking at approaches, which address the problem of multi-tenancy-enabling existing applications helps getting a better feeling regarding the required measures. Guo et al. [GS+07] explore the requirements for multi-tenant applications and present a framework with a set of services to help people designing and implementing multi-tenant applications in an efficient manner. They provide a solution on architectural level to achieve isolation, in particular regarding performance and availability.

Bezemer and Zaidman [BZ10] additionally consider maintenance risks created by multi-tenant applications. Besides costs for future evolution steps, which are not explicitly addressed by the cost model, the discussion also covers recurring maintenance activities such as patches, which are part of the second component of the cost model, the operating costs. The authors describe the need to separate multi-tenant components from single-tenant logic and provide a conceptual architecture blueprint to reduce the risk of high maintenance efforts. This blueprint distinguishes authentication-, configuration- and database-components. In [BZ+10] the results from [BZ10] be applied to evolve an existing single-tenant application to a multi-tenant one with low efforts.

A major factor influencing the operating costs is the actual resource consumption. Kwok and Mohindra [KM08] provide an approach for calculating the overall resource demand for a multi-tenant application instance. Based on this, they propose a tool for optimizing tenant placements, which may consider various constraints defined by SLAs. This helps to significantly reduce the overall resource consumption and therefore has to be taken into consideration when calculating the operating costs. In a similar way, Westermann and Momm [WM10] placed tenants in a cost-minimal way with help of genetic algorithms, but using more detailed resource consumption / performance models based on the software performance curve approach described in [WH10]. Fehling et al. [FLM10] identify several optimization opportunities originating from an intelligent distribution of users among functionally equal resources with varying QoS.

An inappropriate tenant placement / resource optimization not only results in resource wasting, but may also lead to SLA violations causing additional operating costs. In a similar way, missing performance isolation between tenants can easily lead to costly SLA violations. To avoid these situations, Cheng et al. [CSL09] introduce a framework for monitoring and controlling tenant performance using an SLA aware scheduling. In this way it is possible to verify tenant allocations at runtime. This helps to achieve performance isolation between tenants.

Conclusion

In this paper, we introduced a simple cost model for evaluating different options for implementing multi-tenancy depending on the application at hand, the cost for operation / lifecycle management, the expected number of tenants and the expected period of use. Most of the components of this cost function can be estimated using standard effort estimation approaches used in software engineering. To support the estimation of the initial reengineering effort we furthermore pointed out common modifications necessary to implement die different multi-tenancy options while fulfilling the customer requirements.

However, the complexity of such a multi-tenancy or in general SaaS enablement can probably be significantly reduced by developing adequate migration and reengineering techniques, just like in case of service-oriented architectures. At the same time, existing middleware and programming models should be enhanced to natively support multi-tenancy. This would considerably improve the development of new applications for the SaaS model and create new options for migrating existing applications.

References

[BB99] Bengtsson, P.; Bosch, J.: Architecture level prediction of software maintenance. In: Proc. of the Third European Conference on Software Maintenance and Reengineering, 1999, pp. 139–147

[BL+04] Bengtsson, P.; Lassing, N.; Bosch, J.; van Vliet,H.: Architecture-level modifiability analysis (ALMA), Journal of Systems and Software 69(1-2), 2004, pp. 129 – 147.

[BD+03] Benediktsson, O.; Dalcher, D.; Reed, K.; Woodman, M.: COCOMO-Based Effort Estimation for Iterative and Incremental Software Development, Software Quality Control 11(4),2003, pp. 265-281..

[BYV08] Buyya, R.; Yeo, C. S.; Venugopal, S.: Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities. In: Proc. of the 10th IEEE International Conference on High Performance Computing and Communications, 2008.

[BZ10] Bezemer C.; Zaidman, A.: Multi-Tenant SaaS Applications: Maintance Dream or Nightmare. In: Technical Report Series of Delft University of Technology (Software Engineering Group), 2010, Report TUD-SERG-2010-031

[BZ+10] Bezemer, C.; Zaidman, A.; Platzbeecke B.; Hurkmans T.; Hart A.: Enabling Multi-Tenancy: An Industrial Experience Report. In: Technical Report Series of Delft University of Technology (Software Engineering Group), 2010, Report TUD-SERG-2010-030

[CSL09] Cheng, X.; Shi, Y.; Li Q.: A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on SLA. In: Proc. of Joint Conferences on Pervasive Computing (JCPC) 2009, 2009, pp. 599-604.

[DW07] Dubey,A.; Wagle, D.: Delivering software as a service. In: The McKinsey Quarterly, 2007, pp. 1-12

[FLM10] Fehling, C.; Leymann, F.; Mietzner, R.: A Framework for Optimized Distribution of Tenants in Cloud Applications. In: 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010, pp. 252-259

[GS+07] Guo, C.; Sun W.; Huang Y.; Wang, Z.; Gao B.: A Framework for Native Multi-Tenancy Application Development and Management. In: 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007.

[IFP99] IFPUG. Function Point Counting Practices Manual. International Function Points Users Group: Mequon WI, 1999.

[KM08] Kwok, T.; Mohindra, A.: Resource Calculations with Constraints, and Placement of Tenants and Instances for Multi-tenant SaaS Applications. In: Proc. of Service-Oriented Computing - ICSOC 2008, 2008, pp. 633-648

[OG+09] Osipov, C.; Goldszmidt, G.; Taylor M.; Poddar, I.: Develop and Deploy Multi-Tenant Web-delivered Solutions using IBM middleware: Part 2: Approaches for enabling multi-tenancy. In: IBM's technical Library, 2009, http://www.ibm.com/developerworks/webservices/library/ws-multitenantpart2/index.html

[PB01] Daniel J. Paulish and Len Bass. Architecture-Centric Software Project Management: A Practical Guide. Addison-Wesley Longman Publishing Co., Inc., USA, 2001.

[SS+10] Sripanidkulchai, K.; Sahu,S.; Ruan, Y.; Shaikh, A.; Dorai, C.: Are clouds ready for large distributed applications? In: ACM SIGOPS Operating Systems Review, vol. 44 issue 2, 2010, pp. 18-23

[TYB08] Theilmann, W.; Yahyapour, R.; Butler, J.: Multi-level SLA Management for Service-Oriented Infrastructures. In: Towards a Service-Based Internet, 2008, pp. 324-335

[WH10] Westermann, D.; Happe, J.: Towards performance prediction of large enterprise applications based on systematic measurements. In: Proc. of the Fifteenth International Workshop on Component-Oriented Programming (WCOP) 2010, 2010, pp. 71-78.

[WM10] Westermann, D.; Momm, C.: Using Software Performance Curves for Dependable and Cost-Efficient Service Hosting, In: Proc. of the 2nd International Workshop on the Quality of Service-Oriented Software Systems (QUASOSS '10), 2010.

Traceability Link Evolution with Version Control

Patrick Mukherjee[1], Karsten Saller[2], Aleksandra Kovacevic[1], Kalman Graffi[1], Andy Schürr[2], Ralf Steinmetz[1]

[1]Multimedia Communications Lab, Technische Universität Darmstadt
Rundeturmstr. 10, 64289 Darmstadt, Germany
{patmuk, sandra, graffi, ralf.steinmetz}@KOM.tu-darmstadt.de

[2]Real-Time Systems Lab, Technische Universität Darmstadt
Merckstr. 25, 64289 Darmstadt, Germany
{saller, schuerr}@ES.tu-darmstadt.de

Abstract: Traceability links enable the possibility to trace the evolution of a project from the earliest requirements engineering phase over the design and implementation to the test phases of a project. Having a link between the artifacts resulting from these phases, a requirement can be traced to, e.g., the test report, which tested the code implementing the requirement. However, as the artifacts evolve into new versions the traceability links between these artifacts evolve as well. Concurrent work on the artifacts might result in conflicts which reflects in conflicting versions of the traceability links. Often the artifacts created in different phases of a development process are stored in different repositories. In this paper we present a version control system capable of controlling the evolutionary changes of traceability links which connect artifacts residing in different repositories and resolve the conflicts caused by concurrent work on artifacts.

1 Introduction

Traditionally (in requirements management) traceability links have been used to connect requirements gathered in the first phase of a project to more detailed requirements resulting from the initial requirements [RSPE97, WN94]. DOORS [IBM] is the most frequently used tool to support this task.

However, traceability links can be created between any items, be it an artifact, folder or module, in any version. Traceability links can be used for various additional features in a project's development. For example, let us observe artifacts from different development phases, such as requirement, design, implementation and testing. By using traceability links, we can trace back a test case from the last phase of the development to related artifacts from earlier phases, such as source code or requirement specifications. This feature could be helpful in deciding which tests are to be executed to prove the implementation of a specific requirement, if budget or time constraints limit the executable tests. Traceability links between modules can be used to store dependency relations among projects, which

enables configuration management to choose specific versions. For example, a project could link to another project, indicating the minimal needed version it depends on.

When using traceability links in a dynamic environment like software development, maintaining traceability links faces many challenges. This paper will focus on three main challenges:

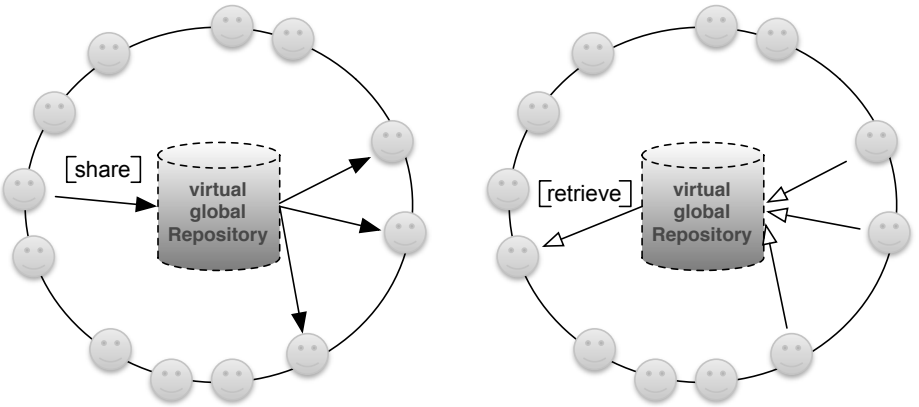
- Traceability links between artifacts should evolve along with the connected artifacts. Like with artifacts this evolution should be tracked, to be able to retrieve the correct state of a traceability link which connects specific versions of artifacts. We present how to overcome this limitation in Section 3.1.
- When storing traceability links in one central place, like most supporting tools do, this place becomes a bottleneck for users which want to access any traceability link, even if the accessed links connect different artifacts. We discuss our approach to overcome this challenge in Section 3.2.
- Often the artifacts created in different phases or by teams in different locations of a development process are stored in different repositories. Our solution to this problem is presented in Section 3.3.
- Concurrent work on connected artifacts might result in conflicts on connecting traceability links. We focus on our solution to this challenge in Section 3.4.

Our approach to maintaining traceability links is integrated in our fully decentralized version control system PlatinVC [Muk10], which is presented in Section 2. Section 3 details how the evolutionary changes of traceability links are handled. We discuss related approaches in Section 4 and close with a conclusion in Section 5.

2 Background: Version Control System PlatinVC

With evolving artifacts in software development a user expects traceability links to be versioned too, to enable all advantages of version control: retrieving an old state of the work to correct mistakes or managing possible conflicts occurred by concurrent work. Version control systems provide the ability to store all changes on artifacts and allow access any instance in the timeline of an artifact later. In this Section we present PlatinVC, a version control system that tracks the evolutionary changes of both, artifacts and the corresponding traceability links between them. In order to understand our approach, we summarize necessary background information about PlatinVC [Muk10].

PlatinVC represents, as a fully decentralized version control system, an opposite approach to the most commonly used client-server based version control systems that rely on a centralized machine to store and manage all artifacts of a project. The obvious drawbacks involved by utilizing an intermediary central server to exchanges and store created versions is having a single point of failure, a central ownership and vulnerability, scalability issues, and increased synchronization as well as communication delays. These drawbacks



(a) retrieve changes (b) share local changes

Figure 1: Sharing and retrieving versions using PlatinVC

are even more emphasized when projects are developed in a globally distributed manner [MKBS08]. Many popular global software projects switched to the emerging distributed version control systems, demonstrating the urgent need for a more suitable version control system. However, distributed version control systems, like, for example, Git [HT] or Mercurial [Mac] lack the global view of centralized systems, where developers do not have to care with whom they exchange their modified artifacts.

The basic features for version control are provided by Mercurial [Mac], the automatic exchange of new versions is carried out by the peer-to-peer middleware implementation FreePastry [FP]. PlatinVCs algorithms care about whom to send and from where to retrieve updates, like presented in Figure 1. Each participating computer, presented as yellow cycle on the circle and called peer, stores the whole repository in form of a mercurial repository. Using the key based routing mechanism [DZD$^+$03] a peer is responsible for any folder, whose hash value calculated using the folders name is similar to the peers identifier (see also [Muk10, MLTS08]. A responsible peer stores the latest versions of all artifacts residing in the folders it is responsible for. If any of these artifacts get updated the resulting snapshot is stored by the responsible peer. When peer retrieves the latest snapshot of all artifacts in a project, PlatinVC delivers them involving the minimum number of responsible peers needed. As the share and retrieve operations work transparent a user experiences the exchange of new versions as if he where communicating with a central repository, called virtual global repository in Figure 1. In this repository several modules can be stored, which consist of artifacts that are versioned using a snapshot. All files are in fact stored by each peer, most likely outdated. The latest snapshots of the artifacts in a folder are stored by its responsible peer, replicated to its direct neighboring peers. Only the peers, whose users retrieved the latest snapshot while no new snapshot was shared, have the latest version of all files.

Despite of its decentralized, peer-to-peer nature, PlatinVC still achieves the complete system view of centralized systems, while overcoming their drawbacks [GKM+07]. All workflows of existing version control systems, centralized or distributed, are supported. PlatinVC even supports a hybrid setup, where other version control systems can interoperate, using PlatinVC as a mediator. Moreover, it introduces an automatic isolation of concurrent work, which separates relevant and required updates from possibly unneeded ones. In this way, branches are only created when necessary. The evolutionary changes of links between files, which can be enhanced by any attributes, are also tracked. That extends snapshot-based version control to other purposes, e.g. traceability of software artifacts. PlatinVC is an alternative to current version control systems, as industry proven components for the most critical parts of the system were reused, and own developed parts where evaluated following [KKM+07].

3 Versioning and Maintaining Traceability Links

Enabling evolution of traceability links rise many questions, which we answer in this Section:

1. How to store them? (Section 3.1)
2. Where to store them? (Section 3.2)
3. What to do when artifacts created in different phases of a development process are stored in different repositories? (Section 3.3)
4. What to do when the linked artifacts change, how to version traceability links and how to resolve concurrent updates on linked artifacts? (Section 3.4)

Answering these questions, we present our approach to the evolution of traceability links in the fully decentralized version control system PlatinVC.

3.1 Structure and Form of Traceability Links

A traceability link can be created between any artifacts, folders or modules. A traceability link in PlatinVC can connect any number of items. For each connected item a companion file is created. It has a generic name formed by the "." prefix (to hide the file in unix-like operating systems), the linked item's name and the suffix "<-linkfile". This companion file is stored along with the corresponding artifact, or, in the case the item is a folder, in the folder and in the case it is a linked module, in the module's topmost folder. The companion file contains the names of *links*, which includes the actual information stored by a traceability link. These entries express that a link exists and serve as an index to find the link documents. No further information is stored here. Thus we avoid updating multiple files if a traceability link is changed.

```
<?xml version="1.0" encoding="UTF-8"?>
<linkType>DesignToCode</linkType>
<status>active</status>
<links>
(...)
        <link partner="A">
                <validFrom>3.4</validFrom>
                <direction>from</direction>
                <relation>derived</relation>
                (...)
                <otherAttribute>something</otherAttribute>
        </link>
        <link partner="B">
                <validFrom>1.8</validFrom>
                <direction>to</direction>
                <relation>derived</relation>
                (...)
                <otherAttribute>something</otherAttribute>
        </link>
(...)
</links>
```

Listing 1: Exemplary traceability link file *A<-link->B*

For each traceability link a link document is created, named with a arbitrary name.

If, for example, an artifact named *A* is connected with a traceability link to an artifact named *B*, there exist the following files: In the same folder where *A* is stored a companion file named *.A<-linkfile* is created. it contains the sole entry "A<-link->B". Next to the artifact *B* a file *.B<-linkfile* is created with the same sole entry. All information of the traceability link are stored in the file *A<-link->B*. Please note that the name can be any name, randomly created, created after a fixed scheme or assigned by a user.

The link document (*A<-link->B* in the previous example) is stored in a folder with the identical name (e.g. *A<-link->B* / *A<-link->B*), which is stored in a folder named ".linkdocuments". This extra folder (with the identical name) is necessary to distribute the link documents among different responsible peers, as they are responsible for files in different folders, like explained above. If the link document connects items, which resides in the same module, this folder resides in the topmost folder of that module. If items of different modules are linked the folder is stored in the topmost folder of a third module, which is named after the modules names in which the linked items are stored in. The name is constructed by the modules names with "<-links->" in between. In the latter case the entry in the item's companion file is prefixed with this name.

An exemplary traceability link (link document *A<-link->B* from the previous example) stored in PlatinVC is presented in Listing 1. A link document has two main parts. The

first xml-tags can specify information about the link, like its type. Here anything related to the traceability link itself, but not specific to any artifact, can be stored. The child xml-tags under <*links*> represent each the end of a link. Here the linked items are specified, by their name, version identifier and branch, if the latter is not the default branch. Any additional attributes belonging to the traceability link and specific to the linked item can be stored here as well.

A traceability link is valid for a specific version of the linked artifacts only. However, for the most projects it can be assumed that it is valid for later versions as well, until the link document is updated to a later version, which states new version identifiers for the connected artifacts. A link document can store any metadata about the connection of the linked items, such as integrity constraints, which can be evaluated to prove whether a connection is justified. This validation, however, is not handled by PlatinVC. The system PlatinVC does not control how links are interpreted, it merely tracks their concurrent evolution.

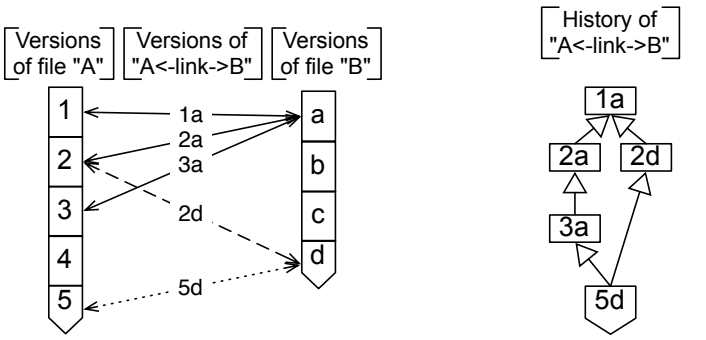
3.2 Distributed storage of traceability links

In opposite to solutions where the whole traceability link information for all files in a project are stored in a central place, the files containing this information (the link documents) are kept in a distributed storage implemented by PlatinVC. Following the key based routing mechanism [DZD+03], the link documents are distributed among different maintaining peers (see also [Muk10]). If two different developers access traceability links, which connect a disjunct set of artifacts, they communicate (most likely) with two different peers. In the rare case, where one peer is responsible for both traceability links and is overloaded by requests, the responsible peer's replicating peers take over.

3.3 Traceability links between different repositories

Often the artifacts created in different phases of a development process are stored in different repositories. This happens, when the staff working in one phase is different from the staff working in another phase, most often because they are employed by different companies. The challenge of maintaining traceability links along with the artifact evolution is even bigger when the repositories belong to different version control systems (e.g. subversion [Pil04], CVS [CVS], Mercurial [Mac]). Thus the requirements might be compiled by a traveling worker, in direct exchange with the customer, managed by a local version control system such as mercurial. The designers of company A create the software architecture, version controlled by CVS, while the developers of company B prefer Subversion for their source code. Lastly, the testers of a different company use their own CVS server to maintain the test reports.

As already explained in Section 2, PlatinVC is based on Mercurial which can use adaptors to many version control systems [O'S09]. Therefore, PlatinVC is capeable to handle



(a) Files and versions

(b) Created history of the link document

Figure 2: Concurrent updates on a linkdocument

several CVS and Subversion repositories at the same time.

3.4 Concurrent updates on linked artifacts

Changing the information stored in a traceability link updates its link document to a new version. Pointing to a different version of a linked artifact changes the link document as well. As a convention a link should only point to more recent versions of the linked artifacts in an update. However, when two different developers first update a linked artifact and update the link document subsequently, without having the latest version created by the other developer, the link's versions cross each other, like depicted in Figure 2(a).

Figure 2(a) illustrates an exemplary situation. The left column represents five versions of artifact "A", the right column four versions of a linked artifact "B", and the arrows in between represents the versions of the linking document (representing a binary traceability link). The version of both artifacts are created by different users. Alice works on "A" and Bob on "B". First Alice creates the artifact "A" and links it to version a of artifact "B". She updates "A" to version 2, updates the link to version 2a, and repeats the same process later to create version 3 of artifact "A" and version 3a of the link document. The versions 1a, 2a, and 3a of the link document are represented with the solid arrow lines. Meanwhile Bob updates artifact "B" to the versions b, c, and d. He updates the link to connect version d of artifact "B" and version 2 of artifact "A", which is the latest version he fetched.

Alice created the version 3 of artifact "A" and updated the link from version 2a to version 3a at the same time when Bob created the versions b, c and d of the artifact "B" and updated the link form version 2a to 2d. Hereby a conflict in the traceability link's version history arose. This conflicting history is stored by PlatinVC in an unnamed branch (as illustrated in Figure 2(b)). Version 3a and 2d are both based on version 2a. This branch can be merged

by pointing to a more recent version of the connected artifacts. Thereby link version 5d can be based on 3a and 2d to merge the diverging history lines. To avoid unnamed branches, updated links should always point to the latest version of the linked artifacts. When the network is seperated into partitions this is not always possible immediately. We integrated the version control aware instant messenger ASKME [MKS08] in order to notify users of a detected conflict.

4 Related Work

Much effort has been spent in the research of traceability links. Most of the research handles the problem of recovering traceability links to identify semantic connections between source-code and documentation [MM03] [ACC+02] [NG06]. None of these approaches can handle the evolution of software projects over time. The links which are identified during the recovery process are valid only for a certain version of the software project. During the development, the software artifacts are changing constantly which is why simple methods for link recovery are not efficiently applicable.

An approach that is leveraging traceability link recovery methods to provide consistent and correct traceability links in an evolving environment is described in [JNC+08] by Hsin-yi et. al. He describes TLEM, a automatic traceability link evolution management tool, that uses an incremental version of the Latent Semantic Indexing algorithm [yJNCD07] to re-evaluate traceability links after each change in an incremental fashion. The presented method re-uses all available information before a change is being triggered and only evaluates the impact of a change on the traceability. Therefore the costs of a full traceability link recovery is being avoided for every new version.

Another interesting approach to handle traceability links in evolving software projects is ArchTrace [MHW06]. Murta describes how a policy based system of rules can automatically manage traceability links between documentation and the corresponding source-code. Instead of reconstructing the traceability links after a certain amount of changes or time, ArchTrace updates these links after every commit operation from a user. ArchTrace is only capeable of maintaining existing traceability links, which means that they have to be created manually by the developers or by a traceability recovery method. The described approach uses a policy-based infrastructure to determine the according actions during an update of a software artifact. The policy rule-set has to be configured by the developer to be more accurate in the managing of traceability links. A conflict arises if more than one policy or no policy is triggered by an update. This has to be resolved manually by the developer. However, it may happen, depending on the configuration of the policies, that the wrong actions are triggered after an update (e.g. adding a new traceability link to another artifact) without any knowledge of the developer. The current implementation of ArchTrace only supports Subversion as version control system and has problems if the software architecture is branched.

A considerably different approach to handle evolving traceability links is described in [CHCC03] by Cleland-Huang et. al. This approach uses a scalable publish-subscribe

paradigm to handle the traceability links by event-notification. Source-code and documentation are linked by publish-subscribe relationships where updated artifacts take the role of publishers and the depending artifacts take the role of subscribers. When e.g. a requirement changes all depending artifacts are notified by specific notification messages which contain detailed information about each update event to support the update process of depending artifacts. Thus, this approach tries to support the overall project management with a distributed project-wide process guidance system. The owners of an artifact which has been notified by a change event are being supplied with the necessary information to update the corresponding traceability links. Hence, the traceability links have to be maintained manually.

Existing approaches to maintain traceability links lack the possibility to operate in distributed environments and are therefore limited in their usability in globally distributed software development environments, in contrast to PlatinVC which was specifically developed for such environments. Operating in distributed environments also enables PlatinVC to maintain traceability links over several repositories. For example, the requirement documentation may be stored in a CVS repository and the source-code in SVN repository. In this case, the traceability links may be stored in a third, independent repository. By utilizing mercurial, PlatinVC is compatible to all version control systems which are compatible to Mercurial (e.g. Subversion or CVS). Another advantage of PlatinVC compared to the presented approaches, is its simplicity: Manual interaction is only needed, if two traceability links are created in parallel with conflict to each other. Both links are kept separately in two separated versions until the developer resolves this conflict.

5 Conclusion

Traceability links bring many advantages to the software development process and their evolution is important and a challenging task. In this paper we presented an approach to version and maintain traceability links in an even more challenging environment where all communication is fully decentralized.

PlatinVC enables the tracking for the evolutionary changes of traceability links between artifacts in different modules, even when stored by different version control systems. It handles and maintains updates of traceability links and their version history. Concurrent updates are handled automatically and a version history for each traceability link is maintained. However, there is currently no support for creating links automatically or assisted. Links can currently only be created automatically via Piki [MLS08], our peer-to-peer based wiki engine.

In future our solution for tracking the changes of traceability links could be combined with a solution to create and change these links automatically. An approach as described in [Kön08] could be used to create or update links between artifacts (semi-)automatically, which could be stored in a decentralized fashion using PlatinVC. However, as a short term goal we strive to implement a distributed access control similar to the concept presented in [GMM+09] for the peer-to-peer social network *LifeSocial* [GGM+10].

References

[ACC+02] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering Traceability Links between Code and Documentation. *IEEE Trans. Softw. Eng.*, 28:970–983, October 2002.

[CHCC03] Jane Cleland-Huang, Carl K. Chang, and Mark Christensen. Event-Based Traceability for Managing Evolutionary Change. *IEEE Trans. Softw. Eng.*, 29:796–810, September 2003.

[CVS] CVS - Concurrent Versions System. `http://www.nongnu.org/cvs/`.

[DZD+03] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, pages 33–44, February 2003.

[FP] FreePastry. `http://www.freepastry.org/FreePastry/`.

[GGM+10] Kalman Graffi, Christian Groß, Patrick Mukherjee, Aleksandra Kovacevic, and Ralf Steinmetz. LifeSocial.KOM: A P2P-based Platform for Secure Online Social Networks. In *Proceedings of the 10th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 161–162, August 2010.

[GKM+07] Kalman Graffi, Aleksandra Kovacevic, Patrick Mukherjee, Michael Benz, Christof Leng, Dirk Bradler, Julian Schroeder-Bernhardi, and Nicolas Liebau. Peer-to-Peer Forschung - Überblick und Herausforderungen. *it - Information Technology (Methods and Applications of Informatics and Information Technology)*, 49(5):272–279, September 2007.

[GMM+09] Kalman Graffi, Patrick Mukherjee, Burkhard Menges, Daniel Hartung, Aleksandra Kovacevic, and Ralf Steinmetz. Practical Security in P2P-based Social Networks. In *Proceedings of the 34th Annual IEEE Conference on Local Computer Networks (LCN)*, pages 269–272, October 2009.

[HT] Junio Hamano and Linus Torvalds. Git - Fast Version Control System. `http://git-scm.com/`.

[IBM] IBM. Rational DOORS - Requirements Management. `http://www-01.ibm.com/software/awdtools/doors/`.

[JNC+08] Hsin-Yi Jiang, T. N. Nguyen, Ing-Xiang Chen, H. Jaygarl, and C. K. Chang. Incremental Latent Semantic Indexing for Automatic Traceability Link Evolution Management. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE '08, pages 59–68, Washington, DC, USA, 2008. IEEE Computer Society.

[KKM+07] Aleksandra Kovacevic, Sebastian Kaune, Patrick Mukherjee, Nicolas Liebau, and Ralf Steinmetz. Benchmarking Platform for Peer-to-Peer Systems. *it - Information Technology (Methods and Applications of Informatics and Information Technology)*, 49(5):312–319, September 2007.

[Kön08] Alexander Königs. *Model Integration and Transformation – A Triple Graph Grammar-based QVT Implementation*. PhD thesis, Real-Time Systems Lab (ES), Technische Universität Darmstadt, Germany, 2008. Supervisor: Andy Schürr.

[Mac] Matt Mackall. Mercurial, a Distributed SCM. `http://selenic.com/mercurial/`.

[MHW06] Leonardo G. P. Murta, Andr Hoek, and ClÂudia M. L. Werner. ArchTrace: PolicyBased Support for Managing Evolving Architecture-to-Implementation Traceability Links. In *In ASE*, pages 135–144. IEEE, 2006.

[MKBS08] Patrick Mukherjee, Aleksandra Kovacevic, Michael Benz, and Andy Schürr. Towards a Peer-to-Peer Based Global Software Development Environment. In *Proceedings of the Software Engineering Conference SE2008*, pages 204–216, February 2008.

[MKS08] Patrick Mukherjee, Aleksandra Kovacevic, and Andy Schürr. Analysis of the Benefits the Peer-to-Peer Paradigm brings to Distributed Agile Software Development. In *Proceedings of the Software Engineering Conference (SE)*, pages 72—77, Februar 2008.

[MLS08] Patrick Mukherjee, Christof Leng, and Andy Schürr. Piki - A Peer-to-Peer based Wiki Engine. In *Proceedings of the International Conference on Peer-to-Peer Computing (P2P)*, pages 185–186, September 2008.

[MLTS08] Patrick Mukherjee, Christof Leng, Wesley W. Terpstra, and Andy Schürr. Peer-to-Peer based Version Control. In *Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 829–834, December 2008.

[MM03] Andrian Marcus and Jonathan I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 125–135, Washington, DC, USA, 2003. IEEE Computer Society.

[Muk10] Patrick Mukherjee. *A Fully Decentralized, Peer-to-Peer based Version Control System*. PhD thesis, Real-Time Systems Lab (ES), Technische Universität Darmstadt, Germany, December 2010. Supervisor: Andy Schürr.

[NG06] Christian Neumuller and Paul Grunbacher. Automating Software Traceability in Very Small Companies: A Case Study and Lessons Learne. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 145–156, Washington, DC, USA, 2006. IEEE Computer Society.

[O'S09] Bryan O'Sullivan. *Mercurial: The Definitive Guide*. O'Reilly & Associates Inc, 2009.

[Pil04] Michael Pilato. *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[RSPE97] Balasubramaniam Ramesh, Curtis Stubbs, Timothy Powers, and Michael Edwards. Requirements traceability: Theory and practice. *Ann. Softw. Eng.*, 3:397–415, 1997.

[WN94] Robert Watkins and Mark Neal. Why and How of Requirements Tracing. *IEEE Software*, 11(4):104–106, 1994.

[yJNCD07] Hsin yi Jiang, Tien N. Nguyen, Carl K. Chang, and Fei Dong. Traceability Link Evolution Management with Incremental Latent Semantic Indexing. In *Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 01*, COMPSAC '07, pages 309–316, Washington, DC, USA, 2007. IEEE Computer Society.

Komponentenbasierte Entwicklung von Firmware für Sensorsysteme in der industriellen Praxis

Philipp Nenninger[*], Tilo Merlin[+], Dirk John[*], Florian Kantz[*]

[*]ABB AG Forschungszentrum
Wallstadterstr. 59
68526 Ladenburg

[+]ABB Automation Products GmbH
Borsigstraße 2
63755 Alzenau

philipp.nenninger@de.abb.com
tilo.merlin@de.abb.com
dirk.john@de.abb.com
florian.kantz@de.abb.com

Abstract: Um der steigenden Komplexität und Variantenanzahl in der Entwicklung von Transmittern für die industrielle Automatisierungstechnik zu begegnen wurde bei ABB bereits seit 2003 damit begonnen Transmitter mit Hilfe eines komponentenbasierten Ansatzes zu entwickeln. Dieser Artikel gibt einen Überblick über die Hintergründe und Grundlagen des Ansatzes sowie über Erfahrungen, welche bei dieser erfolgreichen Einführung von komponentenbasierter Entwicklung gemacht wurden.

1 Einleitung

Durch verschiedene Trends in der industriellen Automatisierungstechnik wurden einfache Sensoren für Größen wie beispielsweise Temperatur oder Druck im Laufe der vergangenen 20 Jahre deutlich komplexer. Kommunizierte ein einfaches Gerät bis dahin den Messwert über eine einfache Punkt-zu-Punkt Verbindung mit einer 4-20 mA Schleife zu dem angeschlossenen Leitsystem, so sind heute Transmitter mit Feldbusanschluss, aufwendiger und oft mehrsprachiger Benutzerschnittstelle (Human machine interface, HMI) mit mehrzeiliger Anzeige (LCD) sowie vielfältigen Diagnosemöglichkeiten für das Gerät selbst aber auch den angeschlossenen Prozess im Angebot aller namhaften Hersteller zu finden. Ein Großteil dieser Erweiterungen wurde erst durch Implementierung der Funktionalität in Software rentabel.

Die eingangs beschriebenen einfachen Geräte wurden aber nicht vollständig von neueren, komplexen Transmittern ersetzt; vielmehr koexistieren diese Geräte und bedienen verschiedene Teilmärkte. Durch diesen Umstand ist die Zahl der zu pflegenden Varianten deutlich gestiegen, was bereits gegen Ende der 1990er zu Problemen in der Entwicklung und vor allem der Wartung dieser Varianten führte. Im Fall von ABB kam zu dieser Zeit eine aggressive Akquisitionsstrategie hinzu, durch die namhafte Transmitterhersteller wie unter anderem Hartmann & Braun, Elsag Bailey und Fischer & Porter integriert wurden. Durch diese Akquisitionen wurde das Portfolio an Geräten deutlich erweitert und bildete den Grundstock für das Instrumentierungsportfolio, für das ABB bekannt ist.

Da diese Firmen aber jeweils eigene Entwicklungsprozesse und –werkzeuge verwendeten, ergab sich ein sehr heterogenes Bild. So existierten zum Beispiel zeitweise 40 verschiedene Elektronikplattformen, von denen wenige in relevanten Stückzahlen gefertigt wurden. Darüber hinaus ergab sich für den Benutzer keine einheitliche Bedienung verschiedener Geräte, selbst wenn sich die Funktionalität glich, wie beispielsweise die Einstellung der Minimum- und Maximumwerte oder Alarmschwellen.

Vor diesem Hintergrund wurde bereits 2003 mit der Einführung eines komponentenbasierten Entwicklungsprozesses begonnen. Nicht zuletzt aufgrund der in der industriellen Instrumentierung üblichen langen Produktlebenszyklen von typischerweise bis zu 10 Jahren zeigten sich die gewünschten Effekte jedoch nicht sofort. In diesem Artikel wird in Kapitel 2 ein Überblick über typische Instrumentierungsgeräte sowie über den komponentenbasierten Entwicklungsansatz gegeben. Kapitel 3 beschäftigt sich mit den Erfahrungen, die durch den Einsatz des komponentenbasierten Entwicklungsansatzes in Projekten sowie der Nichtverwendung in einem Fall resultierten. Kapitel 4 gibt eine Zusammenfassung sowie einen Ausblick.

2 Komponentenbasierte Entwicklung von Transmittern

2.1 Systembeschreibung der Transmitter

Viele der in Kapitel 1 eingeführten Transmitter sind sich in Grundzügen ähnlich. Ein schematischer Aufbau eines solchen Transmitters ist in Abbildung 1 links zu sehen.

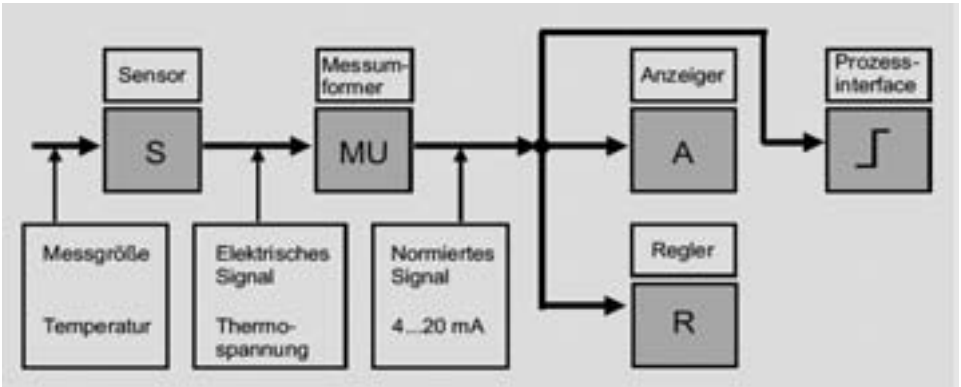


Abbildung 1: Komponenten der Temperaturmesskette (aus [EFG08])

Die eigentliche (physikalische) Messgröße, hier die Temperatur, wird mit Hilfe eines Sensors in ein elektrisches Signal umgewandelt. Typischerweise handelt es sich hier in industriellen Anwendungen um Widerstandsthermometer (Pt100/Pt1000), oder Thermoelemente. Diese ändern ihren Widerstand, beziehungsweise bei Thermoelementen die Ausgangsspannung, in Relation zu der angelegten Temperatur. Dieses elektrische Signal wird von einem Messumformer in ein normiertes Signal umgewandelt, hier 4—20 mA. Hierbei werden Nichtlinearitäten der Sensorelemente ausgeglichen und die Temperatur auf einen vorgegebenen Messbereich abgebildet, der nun in ein Gleichstromsignal überführt werden kann. Anzeiger, Regler und Prozessinterface, Abbildung 1 rechts, sind sowohl als separate Geräte als auch heute teilweise integriert erhältlich.

Diese Bausteine sind in den meisten am Markt erhältlichen Transmittern integriert, wobei je nach Messprinzip auch eine aktive Sensoranregung durch einen Regelkreis notwendig sein kann.

Handelt es sich um ein relativ einfaches Messprinzip, wie beispielsweise im Falle der Temperatur- oder Druckmessung, so kann der Aufbau der Elektronik des Messumformers in etwa der in Abbildung 2 gezeigten Struktur entsprechen.

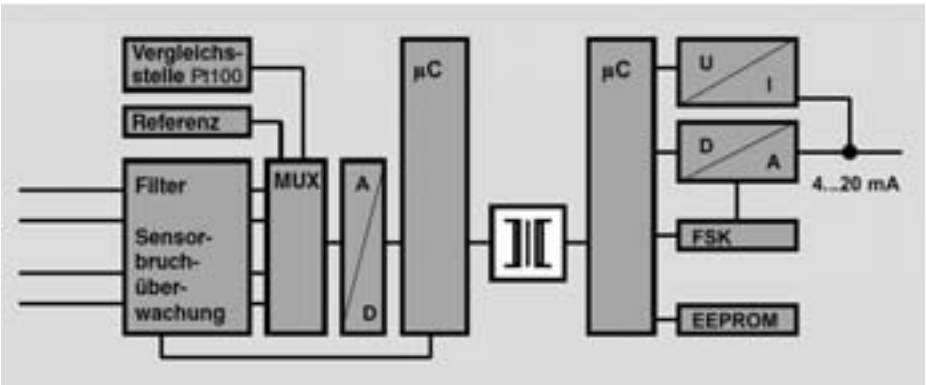


Abbildung 2: Blockschaltbild eines digitalen Transmitters (aus [EFG08])

Auffallend ist die galvanische Trennung der beiden Mikrocontroller (µC) die unter anderem zur Verhinderung von Erdschleifen benötigt wird. In der hier gezeigten Anordnung wird auf beiden Seiten der galvanischen Trennung ein gewisses Maß an „Intelligenz“ in Form eines Mikrocontrollers benötigt.

Eine typische Aufgabenteilung würde beispielsweise Funktionalitäten wie Detektion von Leitungsbrüchen am Sensor und einfache Filterfunktionen auf dem dem Sensor zugewandten Mikrocontroller platzieren, während aufwendigere Signalverarbeitung sowie Kommunikationsprotokolle gegebenenfalls auf dem Mikrocontroller der anderen Seite angesiedelt werden.

2.2 Komponentenbasierte Systementwicklung

Die komponentenbasierte Entwicklung kann in eingebetteten Systemen nur selten von dem Gesamtsystem getrennt betrachtet werden, was auch durch Abbildung 2 deutlich wird. Daher wurde ein Ansatz gewählt, der Systeme in Plattformen gruppiert und in diesen eine höchstmögliche Wiederverwendung von Hard- und Softwarekomponenten anstrebt. Da sich die einzelnen zugekauften Transmitterhersteller oft auf bestimmte Messgrößen spezialisiert hatten, waren Plattformen, wenn sie denn vorhanden waren, entlang dieser Achse organisiert. Im Gegensatz zu der bis dahin üblichen vertikalen Gruppierung nach Messgrößen wurde jedoch eine horizontale Plattformstrategie gewählt, siehe auch Abbildung 3.

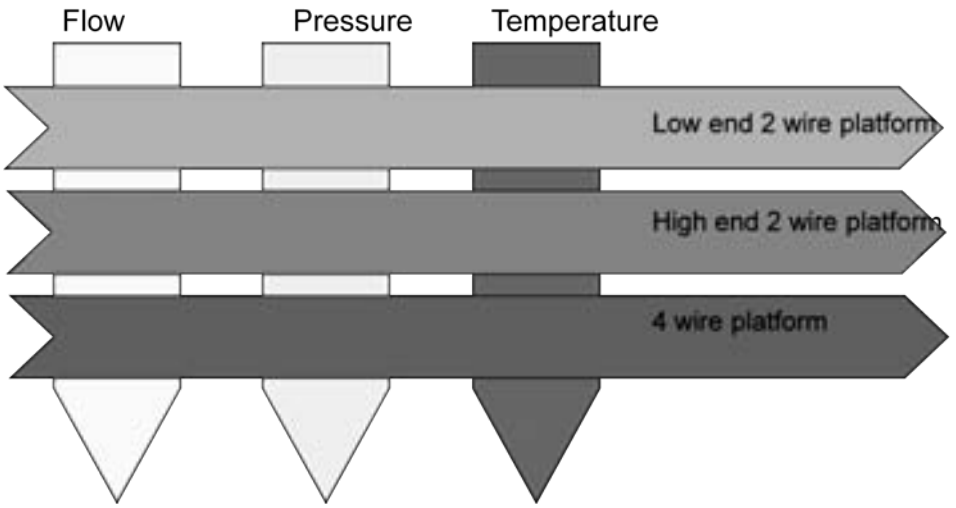


Abbildung 3: Plattformen bilden die Grundlage der komponentenbasierten Entwicklung

Über die Wiederverwendung in einer Plattform hinaus wurde auch die vertikale Wiederverwendung von Komponenten in verschiedenen Produktreihen[1] angestrebt. Hierfür eignen sich beispielsweise Komponenten, die die Linearisierung von Sensorkennlinien implementieren.

Um eine grundlegende Vereinheitlichung der Geräte vornehmen zu können, wurden im ersten Schritt Einschränkungen bezüglich der zu verwendenden Mikrocontroller durchgesetzt. Als Programmiersprache hatte sich C mit kleinen Anteilen Assembler als de-facto Standard durchgesetzt. Mit einem Standardcompiler sowie einem über alle Plattformen gemeinsamen Betriebssystem war also eine gemeinsame Basis vorhanden, auf der die komponentenbasierte Softwareentwicklung fußen konnte.

2.3 Komponentenbasierte Softwareentwicklung

Als Grundkomponente der Softwareentwicklung wurde das sogenannte Subsystem verwendet. Ein Subsystem wurde definiert als eine Gruppe verwandter Funktionen und Daten, die mit Subsystemen anderer Teilen der Software über definierte und bekannte Interfaces kommuniziert. Dies entspricht der gängigen Definition von Komponenten, siehe auch [SVE07].

Die folgenden Abschnitte geben einen Überblick über die Kernpunkte des komponentenbasierten Ansatzes.

[1] Hierbei handelt es sich nicht um „product lines“ im Software-Engineering Sinne.

Vereinheitlichung der Struktur

Um die Wiederverwendung zu vereinfachen, wurden grundlegende Teile der Interfaces standardisiert. Dies ermöglicht auch Entwicklern, die eine Komponente nicht kennen einen rudimentären Umgang mit dieser. Diese Standardisierung kann beispielsweise durch Code-Reviews sichergestellt werden. Um die Akzeptanz bei den beteiligten Entwicklern zu steigern, wurde jedoch auf einen Codegenerator gesetzt, der einerseits die Einheitlichkeit der Interfaces sicherstellt und den Entwickler andererseits von der Arbeit entlastet, eben diese Interfaces, oder wenigstens grundlegende Teile davon, zu implementieren.

Flexibilität und Robustheit

Da die Komponenten mit dem Ziel implementiert werden, sie möglichst unverändert in verschiedenen Produktreihen weiterzuverwenden, stellt sich im Gegensatz zu herkömmlicher Entwicklung die Herausforderung, dass diese auch mit nicht vorhergesehener Benutzung umgehen können. Da Softwareteile bisher relativ aufgaben- und produktspezifisch entwickelt wurden und die Benutzung der Transmitter typischerweise wenige Variationen aufweist, ist dies ein nicht zu unterschätzender Unterschied in dem Entwicklungsansatz für Softwareteile und Softwarekomponenten. Im Unterkapitel 3.1 wird ein drahtloses Gerät als Beispiel für diese unvorhergesehenen Anforderungen näher beschrieben. Die Anforderung der „Robustheit“ schlägt sich in der Implementierung oft durch wiederholtes Überprüfen von Annahmen, besonders in der Umgebung des Interfaces nieder.

Erweiterte Testszenarien

Mit dem Antizipieren neuer Verwendungen geht auch die Erweiterung des Testszenarios einher. Dies wird jedoch durch die Wiederverwendung existierender und damit auch ausführlich getesteter Komponenten mehr als wettgemacht. Dies zahlt sich besonders bei der Entwicklung und Integration neuer Komponenten aus, da existierende, robuste Komponenten als Fehlerquelle (quasi) ausgeschlossen werden können.

Vereinheitlichte Dokumentation

Der Dreh- und Angelpunkt der Wiederverwendung ist die ausführliche Dokumentation der Komponenten und hier besonders der Interfaces. Erst durch gute Dokumentation ist es möglich, funktionierende und getestete Komponenten ohne deutlichen Mehraufwand zu integrieren. Zu diesem Zweck wurde die Dokumentation der Komponenten während des gesamten Entwicklungsprozesses vereinheitlicht. So ist gewährleistet, dass die Verwendung in einem antizipierten Bereich stattfindet und Auswirkungen, beispielsweise auf Laufzeit und Speicherbedarf auch bei Wiederverwendung früh abschätzbar sind.

Die Unterschiede zwischen dem komponentenbasierten Entwicklungsansatz und der herkömmlichen Methode lassen sich also wie folgt zusammenfassen:

- Vereinheitlichung der Struktur, Dokumentation und grundlegender Interfaces
- Design der Komponenten berücksichtigt Wiederverwendung in andersartigen Anwendungen

- Ausführliches Testen, berücksichtigt auch die Robustheit in andersartigen Anwendungen
- Erweiterte und standardisierte Dokumentation

3 Komponentenbasierte Entwicklung in der Praxis

3.1 Wiederverwendung von Komponenten über Produktreihen und Produktgenerationen

Das für den Anwender der nach dem komponentenbasierten Ansatz entwickelten Geräte auffälligste Merkmal ist die Verwendung eines einheitlichen HMIs, siehe auch Abbildungen 4 und 5.



Abbildung 4: Auswahl aus dem Geräteportfolio vor (links) und nach der Entwicklung mit Hilfe des komponentenbasierten Ansatzes

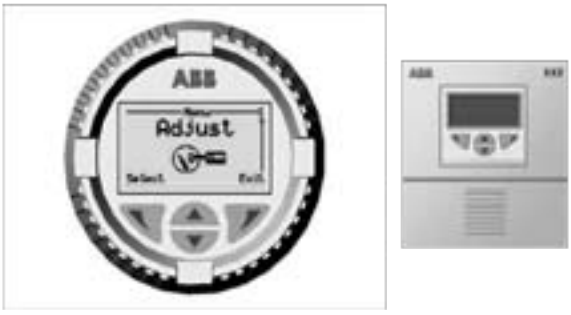


Abbildung 5: Wiederverwendung der HMI Komponente in verschiedenen Produkten

Abbildung 5 zeigt das HMI in zwei verschiedenen Bauformen. Diese Bauformen basieren jedoch intern auf der gleichen Software und ähnlichen Hardwarekomponenten und wurden lediglich im Formfaktor an die Geräte angepasst.

In Tabelle 1 ist ein Vergleich der ersten sechs Projekte, die mit dem komponentenbasierten Ansatz durchgeführt wurden, zu sehen. Dabei ist in der linken Spalte die Anzahl der Varianten aufgeführt, die ohne Zuhilfenahme des komponentenbasierten Ansatzes benötigt worden wären. Die rechte Spalte zeigt die tatsächlich mit dem komponentenbasierten Ansatz entstandenen Varianten.

6 Projekte mit komponenten-basierten Ansatz	**6 Projekte ohne komponenten-basierten Ansatz**
5 HMIs	2 HMIs
5 SW Architekturen	1 SW Architektur
3 Hart HW/SW	1 Hart HW/SW
3 FF & Profibus HW/SW	1 FF & Profibus HW/SW
3 Gerätetreiber Technologien	1 Gerätetreiber Technologie
5 Diagnosesysteme	1 generisches Diagnosesystem
5 Gehäuse	4 Gehäuse
Gesamt = 29	**Gesamt = 11**

Tabelle 1: Vergleich der Varianten in sechs Projekten ohne (links) und mit dem komponentenbasierten Ansatz (rechts)

Besonders hervorzuheben ist hierbei, dass für sechs verschiedene Geräte lediglich eine Softwarearchitektur verwendet wurde. Obwohl dies nicht zwingend durch den komponentenbasierten Ansatz vorgegeben ist, lagen die Vorteile einer Wiederverwendung deutlich auf der Hand.

Bei diesen ersten sechs Projekten handelt es sich um Transmitter, die sich nach Abbildung 3 sowohl in der vertikalen als auch der horizontalen Dimension unterscheiden. Trotz dieser großen Unterschiede ist es dennoch gelungen, sowohl die Kommunikationskomponenten (HART, Foundation Fieldbus (FF) und Profibus) als auch die Diagnose[2] wiederzuverwenden. Allein hier wurde durch den komponentenbasierten Ansatz und die sich daraus ergebende Wiederverwendung die Entwicklung von sechs Komponenten eingespart.

Lediglich bei den HMIs wurden zwei Varianten entwickelt. Der Hauptgrund hierfür war, wie oben erwähnt, der Formfaktor. Nachfolgende Projekte zeigten gerade bei dem HMI jedoch neue Herausforderungen auf. Transmittern werden typischerweise an Stellen im Prozess verbaut, die nicht einfach zugänglich sind, oft auch in für Menschen gefährlichen Zonen. Das HMI wird daher normalerweise nur zur Konfiguration und Inbetriebnahme und seltenen Routinekontrollen verwendet. Die Werte des Transmitters werden am Leitsystem in der Leitwarte überwacht.

[2] Die Diagnosekomponente ist hierbei hauptsächlich für die Diagnose des Gerätes zuständig und macht keine Aussagen über den Zustand des Prozesses.

Andere Geräte wie beispielsweise Regler und Schreiber werden jedoch genau dort installiert und regelmäßig abgelesen. Das HMI spielt bei diesen Geräten daher verständlicherweise eine wichtigere Rolle, die nicht durch das einfache Transmitter-HMI wie in Abbildung 5 gezeigt ausgefüllt werden kann. Das aufwendige HMI eines Schreibers, welches beispielsweise ein 5“ Farb-TFT enthalten kann, ist jedoch für Transmitter ungeeignet. Hier ist also ein one-size-fits-all Ansatz ungeeignet.

In Unterkapitel 2.3 wurde betont, dass Komponenten so entwickelt werden sollten, dass sie auch unter nicht vorhergesehenen Randbedingungen wiederverwendet werden können. Diesem Test wurden einige Komponenten unterzogen, als ein erster Forschungsdemonstrator eines drahtlosen Temperaturtransmitters entwickelt werden sollte. Bisherige Geräte wurden entweder über die 4—20 mA Schleife oder eine separate Energieversorgungsleitung konstant mit Energie versorgt.

Bei drahtlosen Geräten ist aber im Batteriebetrieb entscheidend, dass die Energieaufnahme möglichst gering und die Lebensdauer der Batterie damit möglichst lang ist. Eine einfache Methode, Energie zu sparen, ist es, die Elektronikkomponenten in low-power Zustände zu versetzen und nur aktiv arbeiten zu lassen, wenn gerade ein Messwert benötigt wird. Hierzu wird das System quasi angehalten, ein Zustand, der in bisherigen Geräten nicht vorgesehen war.

Als weitere Herausforderung kam hinzu, dass die meisten Entwickler an dem Projekt weder an Temperaturtransmittern noch mit dem komponentenbasierten Ansatz gearbeitet hatten. Hier stellten sich die einheitliche Dokumentation sowie die einheitlichen Interfaces als deutlicher Vorteil heraus, da die Einarbeitungszeit verkürzt werden konnte. Darüber hinaus verbesserten sie auch die Kommunikation mit den Entwicklern der Komponenten, da eine gemeinsame Wissensbasis vorhanden war.

In diesem Projekt wurde deutlich, dass die mit dem komponentenbasierten Ansatz entwickelten Komponenten die Anforderung an eine Wiederverwendung in nicht vorhergesehenen Anwendungen erfüllen. Da die Komponenten bereits in ihrer ursprünglichen Verwendung getestet waren, wurde der zusätzliche Testumfang deutlich verkleinert.

Das Ergebnis des Projekts ist der in Abbildung 6 zu sehende Forschungsdemonstrator. Neben den oben beschriebenen Merkmalen wurde zusätzlich ein energy-harvesting Modul integriert, welches das Gerät durch die Temperaturdifferenz zwischen Prozess und Umgebungstemperatur speist.

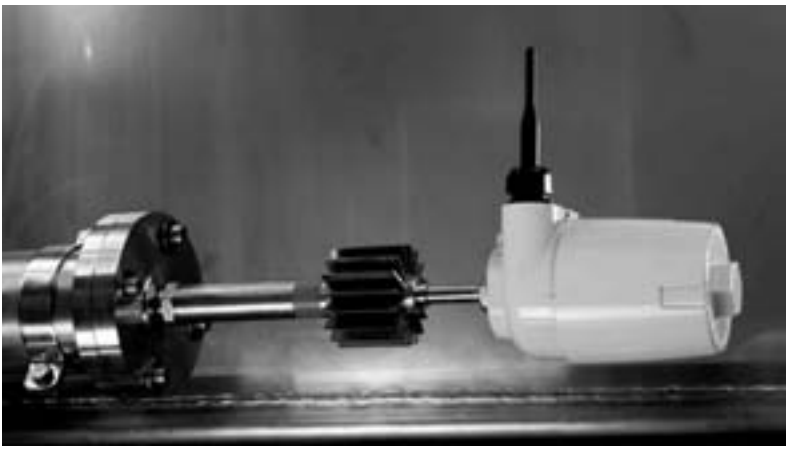


Abbildung 6: Forschungsdemonstrator eines autonomen Temperaturtransmitters, der mit dem komponentenbasierten Ansatz entwickelt wurde

3.2 Projekte außerhalb des komponentenbasierten Entwicklungsansatzes

Trotz der Vorteile des komponentenbasierten Ansatzes wurden auch immer wieder Systeme oder Teilsysteme ohne diesen entwickelt. So wurde beispielsweise ein Mikrocontroller, wie er in Abbildung 2 links zu sehen ist, nicht nach dem komponentenbasierten Ansatz neu entwickelt. Hier wurde die Kommunikation über die galvanische Trennung als Interface verwendet und Wiederverwendung war aufgrund der eingeschränkten Funktionalität quasi ausgeschlossen.

Als bei der Entwicklung eines anderen, ähnlichen Produktes durch eine Verlegung der galvanischen Trennung der zweite Mikrocontroller überflüssig wurde, wurde die Funktionalität in eine Komponente überführt und steht so für weitere Entwicklungen zur Wiederverwendung bereit.

Eine ähnliche Tendenz wurde bei einem Gerät festgestellt, welches von seinem Einsatzzweck her kein Transmitter im herkömmlichen Sinne ist, sondern sich auf bestimmte Aspekte der Kommunikation beschränkt. Auch hier wurde zu Beginn der Entwicklung bewusst auf die Verwendung des komponentenbasierten Ansatzes verzichtet, da eine Wiederverwendung nicht absehbar war. Aufgrund technischer Entwicklungen wurde im Laufe der Zeit eine Wiederverwendung von Teilen der Funktionalität aber opportun, weshalb diese Teile ebenfalls in Komponenten überführt wurden.

Es ist also festzustellen, dass Wiederverwendung, ob vorhergesehen oder nicht, eher die Regel als die Ausnahme darstellt. Dieser Trend wird natürlich durch die durch den komponentenbasierten Ansatz zur Verfügung gestellte Infrastruktur begünstigt.

4 Zusammenfassung und Ausblick

Der in diesem Artikel beschriebene, komponentenbasierte Entwicklungsansatz wurde seit 2003 schrittweise eingeführt, sodass heute bis auf wenige Ausnahmen alle Produkte der entsprechenden Geschäftseinheit komponentenbasiert entwickelt werden. Die Hauptziele, nämlich die Vereinheitlichung von Produkten insbesondere der Benutzerschnittstellen und die Wiederverwendung von bestehenden Komponenten wurden erreicht und teilweise übertroffen. Hierbei existieren Beispiele von Komponenten, die für ihre Wiederverwendung in den komponentenbasierten Ansatz überführt wurden, obwohl sich bei der ursprünglichen Entwicklung explizit dagegen entschieden wurde.

Bei der Wiederverwendung von Komponenten, besonders in Anwendungsfällen, für die diese Komponenten nicht entworfen wurden, ist aber sorgfältig zu prüfen, ob eine direkte Wiederverwendung sinnvoll ist, oder Randbedingungen wie Speicherbedarf oder erforderliche Rechenzeit nicht das neu zu entwerfende System überfordern. In diesem Zusammenhang ist eine Kombination mit einem Produktlinienansatz vielversprechend.

Durch den komponentenbasierten Ansatz ist es aber heute schon möglich, einerseits bei steigender Komplexität die gleichbleibend hohe Qualität der entwickelten Transmitter zu sichern, und andererseits flexibel auf neue Kundenanforderungen zu reagieren. Daher war die Einführung des komponentenbasierten Ansatzes eine für die Steigerung der Wettbewerbsfähigkeit hilfreiche Maßnahme.

Literaturverzeichnis

[EFG08] Ehinger, K.; Flach, D.; Gellrich, L.; Horlebein E.; Huck, R.; Ilgner, H.; Kayser, T.; Müller, H.; Schädlich, H.; Schüssler, A.; Staab, U.: Praxis der industriellen Temperaturmessung, ABB, 2008

[SVE07] Stahl, T.; Völter, M.; Efftinge, S,; Haase, A.: Modelgetriebene Softwareentwicklung, dpunkt.verlag, 2007

SE | 11 SOFTWARE ENGINEERING

IT-Unterstützung von Einsatz- und Rettungskräften: Interdisziplinäre Anforderungsanalyse, Architekturen und Gestaltungskonzepte

Workshop „IT-Unterstützung von Einsatz- und Rettungskräften: Interdisziplinäre Anforderungsanalyse, Architekturen und Gestaltungskonzepte“

Frank Fiedrich[1], Birgitta König-Ries[2], Rainer Koch[3], Volkmar Pipek[4], Stefan Strohschneider[2]

[1] Bergische Universität Wuppertal
fiedrich@uni-wuppertal.de
[2] Friedrich-Schiller-Universität Jena
{birgitta.koenig-ries, stefan.strohschneider}@uni-jena.de
[3] Universität Paderborn
r.koch@cik.uni-paderborn.de
[4] Universität Siegen
volkmar.pipek@uni-siegen.de

Abstract: Die Sicherheitsforschung stellt ein Anwendungsgebiet von IT-Systemen dar, in dem äußerst heterogene Fragegestellungen behandelt und eine Vielzahl unterschiedlicher Stakeholder einbezogen werden müssen. Die Forschungslandschaft wird dabei aktuell gekennzeichnet durch Forschungsprogramme und entsprechende Verbundprojekte, in denen Wissenschaft, Wirtschaft und Endanwender interdisziplinär IT-Unterstützungen für die zivile Gefahrenabwehr entwickeln. Die Beiträge des Workshops adressieren ein breites Spektrum, aus dem die Bereiche „interdisziplinäre Anforderungsanalyse“ und der Entwurf und die Anwendung von „Architekturen und Gestaltungskonzepten“ als zentrale Forschungsbereiche herausgehoben werden.

1 Einleitung

IT-Lösungen in nicht-traditionellen Anwendungsgebieten, wie etwa in der Sicherheitsforschung, erfordern die Bearbeitung äußerst heterogener Fragegestellungen und Einbeziehung aller relevanten Stakeholder. Diese Aspekte zählen auch zu den Leitlinien des vom Bundesministerium für Bildung und Forschung im Rahmen der Hightech-Strategie durchgeführten Programms „Forschung für die zivile Sicherheit“. Gerade im Themenschwerpunkt „Schutz und Rettung von Menschen“ werden hier gemeinsam von Wissenschaft, Wirtschaft und Endanwendern interdisziplinär IT-Unterstützungen für die Menschenrettung aus Gefahrenlagen, den Brand- und Katastrophenschutz entwickelt. Die Interdisziplinarität von Forschungsaktivitäten bedeutet natürlich Chance und Herausforderung zugleich. Dies betrifft insbesondere die Erforschung innovativer IT-Systeme, die den Kern vieler nationaler und internationaler Aktivitäten bilden.

Ein wichtiges Element fast aller Projekte in diesem Bereich ist die explizit als solche benannte nicht-informatorische „Begleitforschung“. Die Kooperation der klassischen technischen Disziplinen und Partnern aus der Wirtschaft wird dabei auf Partner verschiedener Fachgebiete zum interdisziplinären Austausch erweitert. Neben der Psychologie spielen auch andere Fachgebiete wie z. B. die Kommunikations-, Rechts- und Kulturwissenschaft, Human-Factors-Forschung oder auch Pädagogik sowie verschiedene weitere Gebiete der Geistes-, Sozial- und Verhaltenswissenschaft eine wichtige Rolle.

Die Beiträge des Workshops adressieren ein breites Spektrum, aus dem zwei Ziele herausgehoben werden sollen: Auf der einen Seite stellen sie Ansätze zur „interdisziplinären Anforderungsanalyse“ zur Diskussion, die in der Kooperation zwischen unterschiedlichen Fachbereichen entworfen und erprobt wurden. Auf der anderen Seite dokumentieren sie Erfahrungen und Ansätze zum Entwurf und zur Anwendung von Architekturen und Gestaltungskonzepten. Die zivile Sicherheitsforschung wird dabei als Beispiel für Arbeiten und Ergebnisse anwendungsbezogener Forschung betrachtet.

2 Interdisziplinäre Anforderungsanalyse

Allgemeines Ziel ist es, eine bessere Annäherung an den Sachverhalt bzw. eine höhere Annäherung an den Realitätsbereich zu erreichen. Mit Blick auf Produkt- und insbesondere Softwareentwicklungsprozesse ergeben sich inhaltlich und methodisch Potenziale in der Schnittstelle, die besonders für das Requirements Engineering und die Anforderungsanalyse Verbesserungen versprechen. Oftmals treffen hier jedoch unterschiedliche wissenschaftliche Konzepte, Arbeitsmethoden und Forschungsschwerpunkte aufeinander; der Transfer oder die Frage nach der Transferierbarkeit von Ergebnissen der nichttechnischen Partner in die Entwicklung der neuen technischen Systeme und Lösungen gestaltet sich als Herausforderung bei der Projektarbeit.

Schwerpunktthema des Workshops sind daher Fragen, die die Schnittstelle betreffen:

- Wo liegen die Potenziale der interdisziplinären Anforderungsanalyse, welche Risiken müssen berücksichtigt werden?
- Wie kann die potenziell skeptische Einstellung gegenüber zusätzlicher IT konstruktiv durch interdisziplinäre Forschung aufgegriffen werden?
- Wie können Methoden aus verschiedenen Disziplinen aufeinander abgestimmt werden?
- Welche Werkzeuge sind zur anforderungsbezogenen Kommunikation geeignet?
- Wie groß sind die inhaltlichen und methodischen Überschneidungen und Entfernungen?
- Wie können fundierte Lessons Learned und Best Practices erarbeitet werden?

Diese Fragen können provokant auf ihren Kern zurückgeführt werden: Fehlt der Bedarf oder fehlt die Lösung? Wie kommt die Expertise aus den Köpfen der „Begleitforscher“, in die Köpfe der „Techniker“ und zurück?

Die einzelnen Beiträge des Workshops repräsentieren methodische Ansätze um einzelne Fragestellungen zu beantworten. Der Ansatz „FrienDS“ koppelt die „technische“ mit der „sozialen“ Sichtweise und adressiert damit den Bedarf einer Kommunikationsbrücke zwischen dem Softwareentwickler und dem Anwender mit seinen informellen und oftmals impliziten Anforderungen. Auf Basis des Fallbeispiels „LAGE“ diskutiert ein zweiter Artikel einen szenariobasierten Ansatz, um durch die Verknüpfung komplementärer Methoden aus Psychologie und Ingenieurwissenschaft die wechselseitige Nutzung von Inhalten zu ermöglichen und damit iterativ ein gemeinsames Forschungsergebnis zu erreichen. Diese Arbeiten aus der Sicherheitsforschung werden durch eine dritte Arbeit ergänzt, die eine interdisziplinäre, branchenspezifische Methodik zur Produktentwicklung in der elektromechanischen Industrie entwickelt und vorstellt.

3 Architekturen und Gestaltungskonzepte

Entwicklung und Nutzung von IT-Systemen zur Unterstützung aller beteiligten Akteure in Szenarien der zivilen Sicherheit („alltägliche“ zivile Gefahrenabwehr, Bevölkerungsschutz, Katastrophenhilfe, etc.) unterliegen besonderen Rahmenbedingungen durch ihre Öffentlichkeitswirksamkeit und Notwendigkeiten der Verlässlichkeit, Anpassbarkeit/Flexibilität, Sicherheitsrelevanz, Nachvollziehbarkeit und Interoperabilität solcher IT-Systeme. Neben dem besonderen Fokus auf die Anforderungsanalyse ist es daher Ziel des Workshops, Designer und Systementwickler aus diesem Bereich zusammenzubringen, um Erfahrungen und Ansätze für Architekturen, Anwendungen und Vorgehensweisen zu diskutieren. Die Innovativität der diskutierten Ansätze kann dabei sowohl auf der besonders gelungenen Umsetzung der Rahmenbedingungen als auch auf der Erschließung und Einbindung neuer technologischer Konzepte basieren.

Die Themen des Workshops werden daher um folgende Aspekte erweitert:

- Architekturentwürfe für Systeme zur Unterstützung der zivilen Sicherheit
- Domänenspezifische Modelle
- Verallgemeinerbare Randbedingungen bzgl. Systemkonzepten
- Integration von Legacy-Systemen
- Interoperabilität und Standardisierung von Schnittstellen
- Entwicklungs– und Evaluationsmethoden für IT-Anwendungen im Bereich der zivilen Sicherheit
- Methoden und Architekturen zur Einbindung mobiler/ smart Devices

- Einbindung von Location-based Systems und Konzepten/ Technologien des Ubiquitous Computings

- Nutzung von semantischen Technologien in Architekturen

Dieser Themenkomplex wird im Rahmen des Workshops durch zwei Beiträge aus Projekten der deutschen Sicherheitsforschung behandelt. Zum einen wird die Brücke von der Anforderungsanalyse geschlagen, indem Kernanforderungen an Führungsunterstützungssysteme identifiziert werden und daraus ein Vorschlag für eine modulare, serviceorientierte und auf semantischen Technologien beruhende Architektur abgeleitet wird. Zum anderen wird dabei ein neues Konzept interaktiver SOPs vorgestellt. Diese sollen einen standardisierten Einsatzablauf sicherstellen, ohne der Führungskraft den notwendigen Freiraum für eigene Entscheidungen zu nehmen, um so eine optimale Entscheidungsunterstützung in zeitkritischen Situationen zu realisieren.

Organisation

Prof. Dr.-Ing. Frank Fiedrich (Bergische Universität Wuppertal)
Prof. Dr. Birgitta König-Ries (Friedrich-Schiller-Universität Jena)
Prof. Dr.-Ing. Rainer Koch (Universität Paderborn)
Prof. Dr. Volkmar Pipek (Universität Siegen)
Prof. Dr. Stefan Strohschneider (Friedrich-Schiller-Universität Jena)

Programmkomitee

Benedikt Birkhäuser (Bergische Universität Wuppertal)
Prof. Dr. Lothar Cerný (FH Köln)
Tayfur Coskun (TU München)
Dr. Wolf Engelbach (Fraunhofer IAO)
Prof. Dr. Christian Erfurth (FH Jena)
Prof. Dr.-Ing. Jürgen Gerlach (Bergische Universität Wuppertal)
Dr. Gesine Hofinger (Friedrich-Schiller-Universität Jena)
Anna Maria Japs (Universität Paderborn)
Jan-Paul Leuteritz (Fraunhofer IAO)
Ringo Liebscher (Fraunhofer IVI)
Prof. Dr. Max Mühlhäuser (TU Darmstadt)
Dr. Karsten Nebe (C-LAB)
Christian Neuhaus (Universität Siegen)
Jens Pottebaum (Universität Paderborn)
Jun.-Prof. Dr. Kerstin Röse (TU Kaiserslautern)
Carsten Wirth (TU Berlin)
Fabian Wucholt (Friedrich-Schiller-Universität Jena)
Tom Zentek (Forschungszentrum Informatik Karlsruhe)
Dr. Thomas Ziegert (SAP Research CEC)

FRIENDS – eine Kommunikationsbrücke zwischen informatikfremden Fachexperten und Softwareentwicklern in der Anforderungsermittlung

Ivonne Erfurth und Kathrin Kirchner

Friedrich Schiller Universität Jena, Deutschland
Ivonne.Erfurth@uni-jena.de, Kathrin.Kirchner@uni-jena.de

Abstract: Bei der Kommunikation mit dem Auftraggeber über Anforderungen an eine zukünftige Software treten erfahrungsgemäß die meisten Probleme auf. Dies identifiziert den Bedarf einer Kommunikationsbrücke zwischen dem Auftraggeber mit seinen informellen und oftmals impliziten Anforderungen und dem Softwareentwickler, der eine hinreichend formale und eindeutige Spezifikation benötigt. Die Basis zum Aufbau eines gemeinsamen Verständnisses des Problemraums für alle Projektbeteiligten ist eine Kopplung der ‚technischen' mit der ‚sozialen' Sichtweise. Der in diesem Beitrag vorgestellte Ansatz FRIENDS integriert interdisziplinäre Ansätze, um den Prozess der Anforderungsermittlung zu unterstützen und präzise Spezifikationen abzuleiten.

1 Motivation

Mit den herkömmlichen abstrakten Methoden und Techniken der Anforderungsanalyse, die eine präzise und eindeutige Definition für eine Software liefern sollen, sind Auftraggeber oftmals nicht vertraut. Entwickler wünschen und brauchen jedoch eine präzise Spezifikation der Systemstruktur und des Systemverhaltens, um die Anforderungen des Auftraggebers in der zu entwickelnden Software umsetzen zu können. Um diese Kluft zwischen Entwickler und Auftraggeber zu überbrücken, müssen Modellierungsmethoden eingesetzt werden, die beiden als Kommunikationsbrücke dienen. Cheng und Atlee [CA07] geben an, dass die Ermittlung der Anforderungen des Auftraggebers ein gemeinsames mentales Modell voraussetzt. Dieses Modell hilft dem Entwickler, die Anforderungen zu verstehen und ermöglicht damit erst eine erfolgreiche Softwareentwicklung. Aus diesem Ansporn heraus wurde ein auch für informatikfremde Auftraggeber verständlicher Ansatz zur Visualisierung von kundenspezifischen Prozessen – FRIENDS (Friendly Development of Systems) – entwickelt. Grundlage bildeten Erfahrungen, die in verschiedenen Kooperationsprojekten mit lokalen Unternehmen bei der kundenspezifischen Anforderungsermittlung gesammelt wurden. Beispielhaft soll in dieser Arbeit ein Anwendungsszenario aus der Energiewirtschaft herangezogen werden.

1.1 Anwendungsszenario Störfallmanagement

Durch die Regulierung des Strom- und Gasmarktes wurden die Rahmenbedingungen für die Energiewirtschaft neu gesetzt. Durch steigenden Wettbewerb und damit Einsparungen beim Netzbau und der Instandhaltung wird es künftig mehr Störungen als heute geben. Gasnetzbetreiber sind jedoch durch das Energiewirtschaftsgesetz [EnW05] verpflichtet, ein sicheres, zuverlässiges und leistungsfähiges Gasversorgungsnetz zu betreiben.

Das Störfallmanagement von Gasleitungen in einem Stadtwerk stellt somit einen wichtigen Prozess dar, der durch Software unterstützt werden soll [Kir09]: Geht ein Anruf in der Leitstelle ein, dass Gas aus einer Leitung austritt, muss ein Entstörfahrzeug innerhalb von 30 Minuten vor Ort sein. Handelt es sich zum Beispiel um einen Gasaustritt in einem Gebäude, muss zunächst die Gaskonzentration gemessen werden. Dies entscheidet auch darüber, ob das Haus evakuiert werden soll. Die Störstelle muss gefunden und abgedichtet sowie der Hauseigentümer benachrichtigt werden.

Obwohl der Entstördienst nach DVGW Vorschrift [DVG00] geregelt ist, müssen die Feinheiten des Prozesses doch zunächst durch den Entwickler aufgenommen werden. Hierbei ergeben sich einerseits technische Schwierigkeiten, andererseits auch soziale Probleme:

- Die Entwickler sind mit der Fachdomäne weniger vertraut.
- Ältere Mitarbeiter befürchten, dass ihre Arbeit durch die Software ersetzt wird.
- Die Mitarbeiter misstrauen der späteren Softwarelösung, weil nicht alle Details des Prozesses abgebildet wurden.
- Es besteht die Gefahr, dass sich die Mitarbeiter zu sehr auf die Software verlassen und eventuell falsche Entscheidungen des Systems nicht korrigieren.

1.2 Herausforderungen bei der Anforderungsanalyse

Eine Herausforderung bei der Anforderungsermittlung ist, dass Auftraggeber ihre eigenen Anforderungen oftmals nicht konsequent und explizit formulieren können [NE00]. Das Problem verstärkt sich dadurch, dass Softwareentwickler typischerweise weniger Wissen in der Anwenderdomäne besitzen als die jeweiligen Fachexperten. Allerdings benötigen Softwareentwickler ein Verständnis für die Anwenderdomäne [Sha05]. Zwar sind diese fachspezifischen Informationen teilweise aus firmeninternen Dokumenten oder anderer Literatur entnehmbar, wie z. B. beim Störfallmanagement aus den Vorschriften des DVGW. Wenn jedoch keine Dokumente verfügbar sind oder die praktischen Erfahrungen von den Standarddokumenten abweichen, erfolgt die Wissenserfassung über Gespräche mit dem Fachexperten [Sha05]. Diese sind in ihren täglichen Arbeitsabläufen oder Tätigkeiten so routiniert, dass sie oftmals nicht ausdrücken können, wie sie etwas ausführen oder warum. Hinzu kommt, dass es ihnen schwerfällt, sich in ein nicht greifbares und unvorstellbares Produkt – die Software – hineinzudenken.

Für die Erfassung der Prozesse in dem oben vorgestellten Anwendungszenario wurden Besuche in der Werkstatt und auf der Baustelle durchgeführt, um dort direkt Interviews zu führen und die Arbeiten zu beobachten (ethnografisches Interview [Poh08]; Contextual Interview als Technik des Contextual Inquiry [BH98]). Durch den unmittelbaren Kontakt auf der Baustelle fühlten sich die Mitarbeiter ernst genommen, erläuterten gern die einzelnen Arbeitsschritte und berichteten von Problemen. In ihrer vertrauten Arbeitsumgebung fiel es ihnen leichter, ihren Prozess zu erläutern. Da der Entwickler direkt vor Ort war, konnte er gezielt weitere Fragen zu ihm unbekannten Werkzeugen oder Vorgehensweisen stellen, die er dort sah. Dabei konnten auch Fachbegriffe geklärt und Wünsche der Mitarbeiter zur Prozessverbesserung notiert werden.

Eine im nächsten Schritt zu erstellende präzise Spezifikation der zukünftigen Software, die die erfassten Arbeitsabläufe stark verallgemeinert und abstrakt darstellt, trägt weniger dazu bei, diese Anforderungen gemeinsam mit Mitarbeitern der Stadtwerke weiter zu formulieren, zu verfeinern oder zu validieren. Daher ist es unumgänglich, Modellierungsmethoden und Techniken vorzuschlagen, die neben einer präzisen Spezifikation eine positive Atmosphäre zwischen Auftraggeber- und Entwicklerteam erzeugen und auf eine kreative Art und Weise dazu beitragen, die Anforderungen zu ermitteln [MS05]. Diese Techniken könnten ebenfalls zum Einsatz kommen, wo ein ethnografisches Interview schwer durchzuführen ist, wie beispielsweise bei Feuerwehr- und Rettungseinsätzen.

Zur gemeinsamen Diskussion der vom Entwickler erfassten Teilprozesse des Störfallmanagements mit den am Prozess beteiligten Mitarbeitern wurde keine softwaretechnische Spezifikation verwendet. Da die Mitarbeiter nicht mit typischen Modellierungssprachen der Softwaretechnik zur Darstellung von Prozessen vertraut waren, wurde der Prozessverlauf mit Bildern, Texten und verbindenden Pfeilen dargestellt. Diese Visualisierung ist für IT-fremde Personen verständlich. Hier konnten noch einmal kleinere Änderungen vorgenommen und auch schon Ideen zur Verbesserung des Prozesses mit der zu entwickelnden Software diskutiert werden. Für den Entwickler ergab sich jedoch zusätzliche Arbeit, da er die informelle Beschreibung des Prozesses manuell in eine formale Darstellung überführen musste, die für eine spätere Umsetzung in die Software erforderlich war. Eine verständliche Modellierungsmöglichkeit, die in verschiedenen Anwendungsdomänen einsetzbar ist, wäre wünschenswert gewesen.

Um Vorgehensweisen, Modellierungsmethoden und Techniken zu finden oder zu entwickeln, die die Kommunikation zwischen Auftraggeber- und Entwicklerteam unterstützen, den Menschen einbeziehen und die Anforderungen verständlich visualisieren, lohnt ein Blick zu informatikfremden Forschungsrichtungen. Forschungsbereiche, beispielsweise der Kognitions- und Sozialwissenschaften, stellen ein theoretisches Fundament und praktische Techniken für die Ermittlung und Modellierung der Anforderungen bereit [NE00].

2 Der Weg zu einer interdisziplinären Anforderungsanalyse

Um die derzeit existierende Spannung zwischen den informellen Anforderungen des Auftraggebers und einer im Software Engineering gewünschten präzisen Spezifikation aufzu-

brechen, flossen in den hier vorgestellten FRIENDS-Ansatz neben softwaretechnischen Erkenntnissen Forschungsergebnisse informatikfremder Wissensgebiete ein (Abbildung 1).

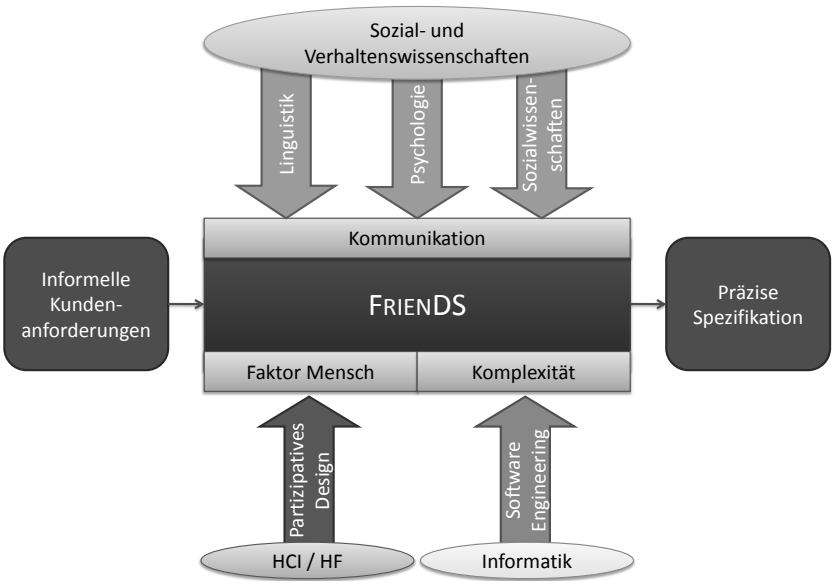


Abbildung 1: Einfließende Forschungsgebiete

Das Forschungsgebiet des Software Engineerings liefert bereits exzellente Vorgehensweisen, Modellierungmethoden und Techniken. Diese münden in eine formale und präzise Spezifikation und machen somit die hohe Komplexität einer Software beherrschbar – ‚technische Sichtweise' (siehe Abbildung 1, Komplexität). Alternative Ansätze aus dem Software Engineering, zum Beispiel agile Methoden, zerlegen komplexe Probleme in kleinere, beherrschbare Teile und ermöglichen eine stärkere Einbeziehung des Fachexperten [BT03]. Für größere Zusammenhänge ist jedoch keine Modellierung vorgesehen. Ambler adaptiert zwar agile Prinzipien auf die Modellierung (agile modeling), schlägt aber keine grafische Darstellung vor [Amb10].

Ziel von FRIENDS ist es nicht, diese exzellenten Verfahren zu ersetzen, sondern sich die Expertise dieser zu Nutze zu machen. Verfahren zur präzisen Anforderungsermittlung und Modellierung sollen geeignet adaptiert werden, so dass sie für informatikfremde Anwender verständlich sind. Ebenso soll bei der Entwicklung von FRIENDS die Anbindung an bereits existierende monumentale Vorgehensweisen und agile Methoden Berücksichtigung finden.

Für die Anforderungsermittlung mit informatikfremden Fachexperten sind Vorgehensweisen und Notationen zur Verfügung zu stellen, die als Kommunikationsbrücke zwischen Auftraggeber- und Entwicklerteam dienen. Ziel ist es dabei, ein gemeinsames mentales Modell der Anforderungen an die zukünftige Software aufbauen. Diese Brücke soll eine positive Atmosphäre zwischen den Projektbeteiligten erzeugen sowie auf eine kreative Art und Weise dazu beitragen, die Anforderungen zu ermitteln – ‚soziale Sichtweise' (siehe

Abbildung 1, Faktor Mensch). Hierfür können Methoden und Techniken des Partizipativen Designs eingesetzt werden. Diese stellen den Menschen in den Vordergrund der Softwareentwicklung und bringen die Expertise für kreative und verständliche Darstellungsformen und Techniken ein.

Das Partizipative Design [BEK87, BB95, BGK95], dessen Wurzeln im Forschungsgebiet der Human-Computer-Interaction (HCI, siehe [HBC+92]) und der Human Factors (HF, siehe [BSHL08]) zu finden sind, vereint als interdisziplinäres Forschungsgebiet Kenntnisse der Psychologie, der Anthropologie, der Soziologie und der Informatik. Ursprünglich auf Grund von Rationalisierungsmaßnahmen auf Initiative von Gewerkschaften als politisch-demokratischer Ansatz in Skandinavien entstanden, zielt das Partizipative Design auf eine verbesserte und intensive Zusammenarbeit zwischen allen am Projekt beteiligten Personen. Das Partizipative Design stellt im Softwareentwicklungsprozess den Faktor Mensch und die Kommunikation zwischen allen durch die Einführung neuer Software betroffenen Personen in den Vordergrund. Aus diesem Ansatz ist durch diese enge Einbindung der zukünftigen Anwender in den Entwicklungsprozess eine Vielzahl von Techniken und Methoden entstanden. Diese ermitteln in Kooperation mit dem informatikfremden Anwender zwar auf eine kreative Art und Weise dessen Expertenwissen, lassen jedoch oftmals einen für eine hinreichend präzise Spezifikation benötigten Formalismus missen. Durch eine Kopplung geeigneter partizipativer Techniken mit softwaretechnischen Modellierungsmethoden besteht die Möglichkeit, die Qualität der Anforderungsermittlung und -modellierung zu steigern und erfolgreiche Softwareentwicklung zu betreiben.

Beide Forschungsbereiche, Software Engineering und Partizipatives Design, fließen als Grundlage in die Anforderungsermittlung ein und stellen gekoppelt mit Kenntnissen der Sozial- und Verhaltenswissenschaften Techniken zur Verfügung, die einerseits zu einer präzisen Spezifikation führen und andererseits für Fachexperten ohne technische Ausbildung leicht verständlich sind. Aufgrund der Berücksichtigung von Erkenntnissen der Sozial- und Verhaltenswissenschaften erfolgt eine Förderung sozialer Kompetenzen (Abschnitt 4), die zu einer positiven Atmosphäre während der Anforderungsermittlung führen. Durch die Kopplung dieser Wissensbereiche besteht zudem die Möglichkeit ein gemeinsames mentales Modell aufzubauen.

Neben diesen beiden Grundsäulen einer möglichen Kommunikationsbrücke zwischen informellen Anforderungen des Auftraggebers und der gewünschten präzisen Spezifikation in der Softwareentwicklung berücksichtigt FRIENDS weitere informatikfremde Wissensgebiete. Um die Kommunikation zwischen den am Projekt beteiligten Personengruppen des Auftraggeberteams und des Entwicklerteams zu verbessern und eine positive Atmosphäre während des Entwicklungsprozesses zu schaffen, lohnt daher ein Blick in Forschungsergebnisse der Sozial- und Verhaltenswissenschaften. Insbesondere die Expertise der Kommunikationswissenschaft liefert vielversprechende Methoden für die Anforderungsermittlung, z. B. zur Planung und Durchführung von Interviews (Abschnitt 3). Die Linguistik und Psychologie liefern beispielsweise vertiefende Kenntnisse über vom Menschen vorgenommene Kategorisierungen [Lak90, Eck91], die auf softwaretechnische Konstrukte adaptierbar sind und somit in der Anforderungsermittlung Berücksichtigung finden können. Dies ermöglicht, ein geeignetes Abstraktionsniveau für eine Kommunikation zwischen Projektbeteiligten aus verschiedenen Fachgebieten zu finden und eine

Kommunikationsbrücke aufzubauen. Um eine Abbildung von den in der Anforderungsermittlung typischerweise vom Auftraggeber informell formulierten Informationen zu in einer präzisen Spezifikation benötigten hinreichend formalen Anforderungen zu ermöglichen, helfen detaillierte Kenntnisse der Grammatiktheorie (Linguistik), die in FRIENDS ebenfalls eingeflossen sind.

3 Ein qualitätsorientiertes Prozessmodell zur Gestaltung der interdisziplinären Anforderungsanalyse

Das Spektrum der Techniken und Methoden, die während der Anforderungsermittlung zur Anwendung kommen, reicht von Ethnografie- und Observationsmethoden, Dokumentenanalyse bis hin zur Durchführung von Interviews und Workshops (siehe auch [Poh08]). Diese empirischen und qualitativen Methoden sind gut erforscht, sorgfältig ausgearbeitet und vielversprechend für die Anforderungsermittlung kundenindividueller Software. Sie gehören jedoch nicht zur typischen Ausbildung eines Softwareentwicklers. Trotz der oftmals geringen Expertise des Softwareentwicklers in diesen Sozial- und Verhaltensforschungsmethoden erfolgt die Anforderungsermittlung in der Praxis häufig über Interviews und Konversationen mit dem Auftraggeber. Da der Entwickler selten die dazu benötigten sozialwissenschaftlichen und psychologischen Kenntnisse mitbringt, erfolgt die Durchführung der Interviews daher oftmals (im Sinne der Sozial- und Verhaltenswissenschaften) Ad-hoc, ohne adäquate Planung und kritische Reflexion.

Um die Kommunikation zwischen Auftraggeber- und Entwicklerteam zu verbessern sowie den Entwickler bei der Durchführung von Interviews zur Anforderungsermittlung zu unterstützen und eine adäquate Planung zur Verfügung zu stellen, wird in diesem Beitrag ein intervieworientiertes Vorgehen bei der Erstellung des FRIENDS-Ansatzes (Anwendungsebene) vorgeschlagen. Dadurch besteht einerseits die Möglichkeit während der Anforderungsermittlung (durch Interviews) gezielt auf das jeweils mit dem Auftraggeberteam zu eruierende Problem zu fokussieren. Andererseits bringt es die Expertise von Auftraggeberteam und Entwicklerteam zusammen, um ein gemeinsames Verständnis – ein gemeinsames mentales Modell – für die Problemdomäne aufzubauen.

Dieses intervieworientierte Vorgehen kam während der Konzeption und Entwicklung des FRIENDS-Werkzeuges C4U (CUTA for you), einer formalisierten Variante von Lafrenières CUTA (siehe Abschnitt 4), als qualitätsorientiertes Prozessmodell zum Einsatz (Entwicklungsebene). Damit wurde ein Prozessmodell verwendet, das nicht nur auf softwaretechnischen Vorgehensmodellen, sondern auf einem Vorgehen der Sozialwissenschaften zur Planung und Vorbereitung von Interviews basiert. In die Entwicklung flossen daher verschiedene Elemente des problemzentrierten Interviews von Witzel [Wit89] ein.

Die Konzeption und Entwicklung von C4U greift somit für die Entwicklungsebene (Abbildung 2, rechts) das Prozessmodell des problemzentrierten Interviews zur Vorbereitung und Planung von Interviews (Anwendungsebene) auf (Abbildung 2, links). Die Grafik stellt sozusagen die Metaperspektive auf den Entwicklungsprozess von FRIENDS dar.

Im ersten Schritt, der Problemformulierung und Problemanalyse, erfolgt eine Festlegung

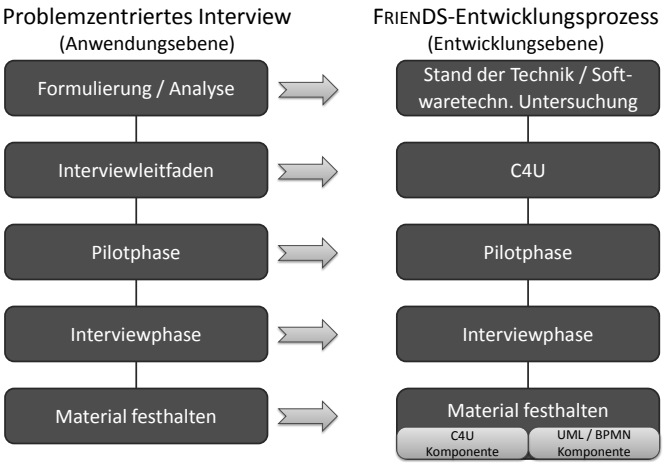


Abbildung 2: Prozessmodell Planung und Durchführung problemzentrierter Interviews versus Prozessmodell zur Entwicklung von C4U

der gewünschten Ergebnisse und Ziele, die das Interview offenlegen soll. Dabei muss im Entwicklungsprozess geklärt werden, welche Methoden für die Entwicklung und Validierung der Anforderungen geeignet sind und was der Entwickler benötigt. Die Analyse von relevanten Konstrukten, um notwendige Informationen beim Anwender zu erfassen, führen direkt zur Erstellung des Interviewleitfadens.

In den Sozialwissenschaften strukturiert ein Interviewleitfaden semi-standardisierte Interviews. Der Leitfaden ist eine Liste von Fragen, die wesentlich in jedem für einen bestimmten Zweck durchgeführten Interview sind. Er stellt sicher, dass der Interviewer alle relevanten Aspekte adressiert und erforderliche Informationen abfragt. Mit dem Ziel, bei der Anforderungsermittlung mit einem Auftraggeberteam ohne technische Ausbildung verallgemeinerte und abstrakte Terminologie zu vermeiden, muss der Softwareentwickler die Fragen zur Anforderungsermittlung an die Sprache des Auftraggeberteams anpassen können. Statt einer Liste ausgearbeiteter Fragen kommt an dieser Stelle das Werkzeug C4U (siehe Abschnitt 4) als Kommunikations- und Visualisierungsmedium zum Einsatz.

Die Pilotphase testet den Interviewleitfaden. Erste Erfahrungen mit einem ausgewählten Personenkreis fließen in diesen ein, gegebenenfalls erfolgt eine Anpassung. Eine erste Evaluation klärt, ob C4U ein verständliches Mittel zur Ermittlung und Visualisierung der Auftraggeberwünsche darstellt [EK10]. Hier treffen Anwendungs- und Entwicklungsebene zusammen.

Die Interviewphase, die das Durchführen der Interviews darstellt, schließt an die Pilotphase an. Dies entspricht im FRIENDS-Prozessmodell der Durchführung der Interviews während der Anforderungsanalyse in realen Softwareprojekten (Anwendungsebene).

In den Sozial- und Verhaltenswissenschaften erfolgt die Dokumentation des Interviews über Video- oder Audioaufnahmen. Nach einer Transkription des Interviews analysieren Experten die Ergebnisse. Für FRIENDS entspricht dies der Transformation der C4U-

Ergebnisse in softwaretechnische Modellierungssprachen (Entwicklungsebene). Damit ist kein Transkript des Interviews notwendig. Eine präzise Spezifikation, zum Beispiel in UML oder BPMN, kann hergeleitet werden.

4 Das Werkzeug: C4U als Interviewleitfaden

Für den Interviewleitfaden wurden partizipative Kartenspiele, wie z. B. PICTIVE [Mul91], TOD [DMK98], CUTA [Laf96], aufgegriffen und derart adaptiert, dass eine Abbildung in abstrakte Modellierungsmethoden der Softwaretechnik möglich ist. Im Fokus dieser Arbeit standen zwei wesentliche Vertreter zur Modellierung von Prozessen. Einerseits die Business Process Modeling Notation (BPMN, [OMG10a]), andererseits Aktivitätsdiagramme der Unified Modeling Language (UML, [OMG10b]). Zur Modellierung von Prozessen mit dem Auftraggeber fiel die Wahl auf Collaborative Users' Task Analysis – CUTA. CUTA, Collaborative Analysis of Requirements and Design, ist ein partizipatives Kartenspiel zur Tätigkeitsanalyse. Auf einfache und praktische Art und Weise analysiert CUTA interne Arbeitsprozesse oder Geschäftsprozesse auf der Ebene des Auftraggebers.

Die im originalen CUTA verwendeten Karten sind einfach gehalten und enthalten neben einem Bild lediglich drei allgemein gehaltene Textfelder für die Aktivität (Activity), die eine Arbeitstätigkeit beschreibt, deren Dauer (Duration) und Häufigkeit (Frequency). Das Bild wird passend zur jeweiligen Aktivität gewählt (siehe Abbildung 3 links). Die Aktivitätsbeschreibung besteht aus einem kurzen Satz in natürlicher Sprache. Die in der Aktivität angegebene Person ist typischerweise absichtlich konkretisiert und nicht anonym. Dies ermöglicht den Anwendern von CUTA eine stärkere Identifikation mit der zu analysierenden Arbeit.

In einer CUTA-Sitzung ermittelt eine Gruppe von nicht mehr als sechs Personen (mindestens 3 Anwender des Auftraggeberteams, ein Entwickler und ein Moderator) Schritt für Schritt die Arbeitsabläufe gemeinsam. Dabei beschreiben die Teilnehmer den Arbeitsprozess anhand der Anordnung der Karten auf einer gemeinsamen Arbeitsfläche. Ähnlich wie in einem Kartenspiel legen die Teilnehmer die Karten auf einen Tisch und bringen diese in eine Reihenfolge. Dabei können sie Karten verschieben, neue Karten in die bereits gelegte Reihenfolge einschieben oder Karten ersetzen.

Durch den spielerischen Charakter erfolgt mit CUTA eine einfache Ermittlung von Prozessen des Auftraggebers beziehungsweise individueller Arbeitsschritte [Laf96]. Neben gesellschaftsfördernden und kommunikationsunterstützenden Eigenschaften von Kartenspielen bietet CUTA weitere Vorteile, die für eine Verwendung als Brückensprache zwischen Entwickler und Anwender sprechen. Dennoch bringt CUTA einige Nachteile mit sich. Beispielsweise kann der Entwickler für den Fall, das zwei Karten nebeneinander liegen wie in Abbildung 3 links, nicht herauslesen, ob es sich um eine Entscheidung (nur eine der beiden Aktivitäten wird ausgeführt) oder um eine Nebenläufigkeit (beide Aktivitäten werden ausgeführt) handelt. An dieser Stelle muss der Entwickler manuell eingreifen, um ein formaleres Modell aus dieser informellen Beschreibung zu erstellen. Dies ist eine potentielle Fehlerquelle. Um diese zu vermeiden, wurden zusätzlich verständliche

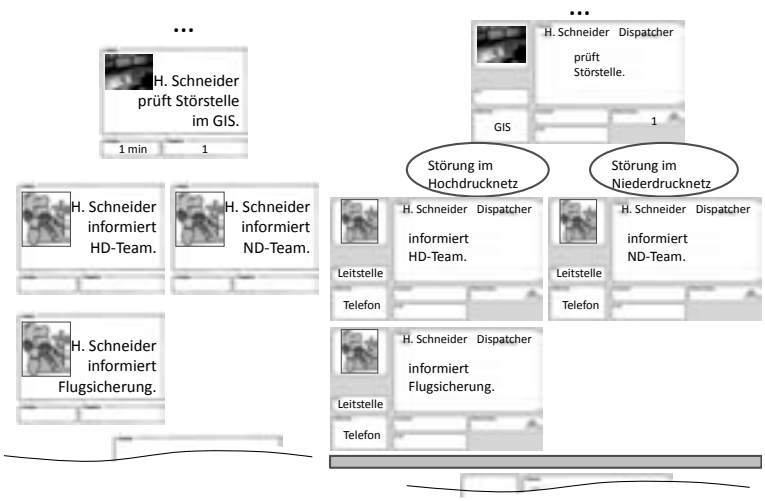


Abbildung 3: Ausschnitt aus einem CUTA-Arbeitsprozess [Laf96] und einem C4U Prozess für das Störstellenmanagement

Prozesselemente in Form neuer Karten eingeführt, um eine eindeutige Unterscheidung vorzunehmen. Ebenso wurden weitere Information auf den Aktivitätskarten aufgenommen, um softwaretechnisch benötigte Informationen zu erfassen [ER10]. In Abbildung 3 ist rechts ein Beispiel eines solchen formalisierten CUTA (C4U) Prozesses aus dem Anwendungsszenario Störfallmanagement zu sehen, bei dem die formalisierten Karten verwendet wurden. Durch den Einsatz von Alternativenkarten ist nun die Entscheidung im Prozessverlauf deutlich zu erkennen.

Die Formalisierung von C4U erlaubt eine Transformation in gebräuchliche Modellierungsprachen, die mit den originalen CUTA-Karten bisher nicht möglich war. Für C4U wurden ein Metamodell erstellt und detaillierte Transformationsregeln für die Umwandlung in UML Aktivitäts- und BPMN-Diagramme auf Basis der zu Grunde liegenden Metamodelle definiert [EK10]. Da eine Karte genau eine Aktivität in einem Prozessablauf beschreibt, kann diese im UML Aktivitätsdiagramm als Aktion und in BPMN als Task übernommen werden. Durch die Einführung von „Kontrollelementen“ (Karten für Alternative und Parallelität) ist eine Abbildung des Kontrollflusses in UML Aktivitätsdiagramm bzw. BPMN möglich.

5 Evaluation

In einer Pilotphase wurden 15 Interviews mit C4U in verschiedenen Anwendungsbereichen wie in der Landwirtschaft [KEM+10], Pharmazie, Universitätsverwaltung und im IT-Bereich erfolgreich angewendet [EK10]. Alle Interviewten waren mehr oder weniger

vertraut im Umgang mit Computern, waren jedoch erstmals in einen Softwareentwicklungsprozess integriert. Sie sind Experten in ihrer Fachdomäne und haben teils langjährige praktische Erfahrung in ihrem Expertengebiet.

Die durchgeführten Interviews wurden bezüglich der Praxistauglichkeit von C4U analysiert. Zusätzlich wurden die Fachexperten nach der Interviewdurchführung befragt, ob die Karten ihnen während des Interviews geholfen haben und ob sie verständlich und nützlich für sie waren. Die in der Pilotphase durchgeführten Interviews zeigten, dass die Karten den Interviewten halfen ihre Geschäftsprozesse zu artikulieren. Alle Experten gaben eine positive Rückmeldung. Dies zeigt, dass die Karten insbesondere für Anwender ohne technische Ausbildung eine große Unterstützung bei der Erläuterung und der Strukturierung ihrer Prozesse sind. Die Anwender aus dem IT-Bereich äußerten sich ebenfalls positiv. Sie nutzten die Karten beispielsweise als Gedächtnishilfe während der Interviews. Es stellte sich heraus, dass die Visualisierung mit C4U einerseits dem Anwender hilft seine Prozesse und Arbeitsschritte zu strukturieren. Andererseits half C4U das Interview zu strukturieren. Die Karten dienten als Leitfaden durch das Interview. Der Interviewer brauchte den Interviewten nicht zu unterbrechen, wenn er gerade im Redefluss war und konnte wichtige Information mittels der Karten dokumentieren. Der Interviewte konnte mit Hilfe der Karten den Prozess verfolgen und den Interviewer in einer aufgeschlossenen und vertrauensvollen Weise korrigieren.

Die Interviews zeigten, dass C4U hilft, Abläufe zu visualisieren und zu klassifizieren. Aus konkreten Fallbeispielen ermöglicht C4U einen Schritt zur Abstraktion: Muster (Organisations- und Handlungsstrukturen) können erkannt werden. Die positiven Erfahrungen in verschiedenen Branchen zeigen, dass eine Anwendung in andere Einsatzgebiete übertragbar ist. Im Konkreten muss die Erfassung der Arbeit von Rettungskräften erprobt werden, um genauere Aussagen zu treffen. Dennoch handelt es sich bei Rettungskräften um Experten, die viel Fachwissen und Erfahrungen mitbringen, während Entwickler mit dieser Domäne wenig vertraut sind. Hier bieten die C4U–Karten eine Kommunikationsgrundlage, die hilft, die typischerweise komplexen Szenarien für alle Beteiligten verständlich darzustellen. Die Karten fördern das Verständnis des Entwicklers für das ihm fremde Fachgebiet.

6 Zusammenfassung

Basierend auf dem Anwendungsszenario Störfallmanagement wurde in dieser Arbeit ein interdisziplinärer Ansatz für die Anforderungsanalyse entwickelt, der die technische und soziale Sichtweise koppelt und auf diese Weise eine Kommunikationsbrücke zwischen informatikfremden Fachexperten und Softwareentwicklern schlägt.

Mit Hilfe des qualitätsorientierten Prozessmodells, das auf das Vorgehen bei der Planung und Durchführung von problemzentrierten Interviews aufbaut, wird bereits während der Konzeption und Entwicklung von C4U das Ziel verfolgt, dem Softwareentwickler ein qualitativ hochwertiges Interview zu ermöglichen. Das als Interviewleitfaden erstellte C4U stellt dem Entwickler ein Kommunikationsmittel zur Verfügung, das ihm hilft, wichtige

und für eine präzise Spezifikation benötigte Informationen zu erfassen und zu dokumentieren. Durch die zusätzliche Berücksichtigung informatikfremder Forschungsbereiche trägt FRIENDS dazu bei, ein gemeinsames mentales Modell zu finden, eine Kommunikationsbrücke aufzubauen und eine positive Atmosphäre zu schaffen sowie die Kreativität zu fördern, damit die Anforderungen so weit wie möglich vollständig und korrekt ermittelbar sind.

Literatur

[Amb10] Scott W. Ambler. Agile Modeling. `http://www.agilemodeling.com/`, 2010.

[BB95] Gro Bjerknes und Tone Bratteteig. User participation and democracy: a discussion of Scandinavian research on systems development. *Scand. J. Inf. Syst.*, 7(1):73–98, 1995.

[BEK87] Gro Bjerknes, Pelle Ehn und Morten Kyng. *Computers and Democracy: a Scandinavian Challenge*. Avebury, 1987.

[BGK95] Susanne Bødker, Kaj Grønbæk und Morten Kyng. Cooperative design: techniques and experiences from the Scandinavian scene. In Ronald M. Baecker, Jonathan Grudin, William A. S. Buxton und Saul Greenberg, Hrsg., *Human-computer interaction: toward the year 2000*, Seiten 215–224. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[BH98] Hugh Beyer und Karen Holtzblatt. *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann, 1998.

[BSHL08] Petra Badke-Schaub, Gesine Hofinger und Kristina Lauche, Hrsg. *Human Factors: Psychologie sicheren Handelns in Risikobranchen*. Springer-Verlag, Berlin, 2008.

[BT03] Barry Boehm und Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Professional, 2003.

[CA07] Betty H. C. Cheng und Joanne M. Atlee. Research Directions in Requirements Engineering. In *FOSE '07: 2007 Future of Software Engineering*, Seiten 285–303, Washington, DC, USA, 2007. IEEE Computer Society.

[DMK98] Tom Dayton, Al McFarland und Joseph Kramer. *User Interface Design, Bridging the Gap from User Requirements to Design*, Kapitel Bridging User Needs to Object Oriented GUI Prototype via Task Object Design, Seiten 15–56. CRC Press LLC, Boca Raton, FL, USA [u.a], 1998.

[DVG00] DVGW. *G 456-3. Beurteilungskriterien von Leckstellen an erdverlegten und freiliegenden Gasleitungen in Gasrohrnetzen*. Wirtschafts- und Verlagsgesellschaft, Bonn, 2000.

[Eck91] Thomas Eckes. *Psychologie der Begriffe: Strukturen des Wissens und Prozesse der Kategorisierung*. Habilitationsschrift (1990), Universität des Saarlandes, Göttingen, 1991.

[EK10] Ivonne Erfurth und Kathrin Kirchner. Requirements Elicitation with Adapted CUTA Cards: First Experiences with Business Process Analysis. In *Proceedings of 15th IEEE International Conference on Engineering of Complex Computer Systems*, Seiten 215–223, Marz 2010.

[EnW05] Gesetz über die Elektrizitäts- und Gasversorgung BGB I 2005, 1970 (3621), 2005.

[ER10] Ivonne Erfurth und Wilhelm R. Rossak. CUTA4UML: Bridging the Gap Between Informal and Formal Requirements for Dynamic System Aspects. In *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Seiten 171–174, New York, NY, USA, 2010. ACM.

[HBC+92] Thomas T. Hewett, Ronald Baecker, Stuart Card, Tom Carey, Jean Gasen, Marilyn Mantei, Gary Perlman, Gary Strong und William Verplank. ACM SIGCHI Curricula for Human-Computer Interaction. Bericht, ACM SIGCHI, 1992. Online `http://old.sigchi.org/cdg/`, zuletzt besucht am 10.9.2010.

[KEM+10] Kathrin Kirchner, Ivonne Erfurth, Sarah Möckel, Tino Gläßer und André Schmidt. A Participatory Approach for Analyzing and Modeling Decision Processes: A Case Study on Cultivation Planning. In Basil Manos, Nikolaos Matsatsinis, Konstantinos Paparrizos und Jason Papathanasiou, Hrsg., *Decision Support Systems in Agriculture, Food and the Environment: Trends, Applications and Advances*, Seiten 138–154. IGI Global, 2010.

[Kir09] Kathrin Kirchner. *Ein räumliches Entscheidungsunterstützungssystem für das Instandhaltungsmanagement in Gasnetzen*. Peter Lang, 2009.

[Laf96] Daniel Lafrenière. CUTA: A Simple, Practical, Low-Cost Approach to Task Analysis. *Interactions*, 3(5):35–39, 1996.

[Lak90] George Lakoff. *Women, Fire, and Dangerous Things. What Categories Reveal about the Mind*, Kapitel Idealized Cognitive Models, Seiten 68–76. University of Chicago Press, Chicago, IL, USA [u.a.], 1990.

[MS05] Ian McClelland und Jane Fulton Suri. Involving people in design. In John R. Wilson und Nigel Corlett, Hrsg., *Evaluation of Human Work*, Seiten 281–333. CRC Press, 2005.

[Mul91] Michael J. Muller. PICTIVE – An Exploration in Participatory Design. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, Seiten 225–231, New York, NY, USA, 1991. ACM Press.

[NE00] Bashar Nuseibeh und Steve Easterbrook. Requirements Engineering: A Roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, Seiten 35–46, New York, NY, USA, 2000. ACM.

[OMG10a] OMG. Business Process Model and Notation (BPMN) Version 2.0. `http://www.omg.org/cgi-bin/doc?dtc/10-06-04.pdf`, 2010.

[OMG10b] OMG. OMG Unified Modeling Language Superstructure. Version 2.3. `http://www.omg.org/spec/UML/2.3/Superstructure`, 2010.

[Poh08] Klaus Pohl. *Requirements Engineering – Grundlagen, Prinzipien, Techniken*. dpunkt.verlag, Heidelberg, 2.. Auflage, 2008.

[Sha05] Nigel Shadbolt. Eliciting expertise. In John R. Wilson und Nigel Corlett, Hrsg., *Evaluation of Human Work*, Seiten 185–218. CRC Press, 2005.

[Wit89] Andreas Witzel. Das problemzentrierte Interview. In Gerd Jüttemann, Hrsg., *Qualitative Forschung in der Psychologie. Grundfragen, Verfahrensweisen, Anwendungsfelder*, Seiten 227–256. Asanger Roland Verlag, Heidelberg, 1989.

Interdisziplinäre Anforderungsanalyse bei der Gestaltung von Systemen zur Erstellung eines gemeinsamen Lagebilds in großen Schadenslagen

Christian Lindemann[1], Jens Pottebaum[1], Gesine Hofinger[2], Laura Künzer[2], Rainer Koch[1]

[1] Computeranwendung und Integration in Konstruktion und Planung (C.I.K.)
Universität Paderborn, Fakultät für Maschinenbau
Pohlweg 47-49
33098 Paderborn
{lindemann, pottebaum, r.koch}@cik.upb.de

[2] Team HF – Hofinger Forschung Beratung Training
Hohenheimer Str. 104
71686 Remseck
{gesine.hofinger, laura.kuenzer}@team-hf.de

Abstract: Großschadenslagen erfordern das Zusammenwirken verschiedener Behörden und Organisationen mit Sicherheitsaufgaben (BOS). Durch ein kollektives Verständnis der Strukturen, Ereignisse und Prozesse im Einsatz kann das Handeln koordiniert und Konflikte vermieden werden. Das Projekt LAGE[1] erarbeitet einen Standard für den Austausch von Meldungen zur Gewährleistung organisatorischer, semantischer und technologischer Interoperabilität. Dazu wurde ein stark interdisziplinärer, szenariobasierter Ansatz gewählt. Die Anforderungsanalyse wird als kontinuierlicher Prozess durch Psychologen und Ingenieure durchgeführt. Dieser Beitrag diskutiert den Ansatz, durch die Verknüpfung komplementärer Methoden in einem gemeinsamen Vorgehensmodell die wechselseitige Nutzung von Inhalten zu ermöglichen und damit iterativ ein gemeinsames Forschungsergebnis zu erreichen.

1 Einleitung

Um große Schadenslagen erfolgreich zu bewältigen, ist ein gemeinsames Lagebild der beteiligten Akteure ein wesentliches Element. „Als Lage bezeichnet man die Gesamtheit aller Gegebenheiten, die sowohl das Schadenereignis (‚Fremde Lage') als auch die Schadenabwehr (‚Eigene Lage') beschreiben. Ort, Zeit, Wetter sind Schadenereignis wie Schadenabwehr gleichermaßen beeinflussende Bedingungen." [Sc98]. Das Lagebild beschreibt eine Sicht auf die zu einem bestimmten Zeitpunkt bekannten Teile der Lage. Das Verbundprojekt LAGE soll es den Führungskräften verschiedener Organisationen ermöglichen, ein solches Lagebild gemeinsam zu pflegen. Dies bedeutet die Erstellung

[1] Das Projekt LAGE wird vom Bundesministerium für Bildung und Forschung gefördert (Fördernr. 13N10585 bis 13N10589). Die Autoren danken den Projektpartnern: Institut für Feuerwehr- und Rettungstechnologie der Stadt Dortmund (IFR), IABG mbH, PSI Transcom GmbH, Unternehmensberatung H & D GmbH und den beteiligten Repräsentanten der Behörden und Organisationen mit Sicherheitsaufgaben der Stadt Dortmund. Die hier beschriebenen Ansichten repräsentieren nicht zwingend die Meinung des Projektverbunds.

eines Konzeptes, das die organisatorische, semantische und technologische Interoperabilität umfasst. Damit ergibt sich eine Überschneidung zwischen personen-, team- und aufgabenbezogenen Schwerpunkten (insbesondere organisatorische und semantische Interoperabilität) sowie struktur-, prozess- und technikbezogenen Aspekten (semantische und technologische Interoperabilität). Diese müssen interdisziplinär aus psychologischer und ingenieurwissenschaftlicher Perspektive betrachtet werden, so dass sich die Frage ergibt: Wie können methodisch unterschiedliche Ansätze der Psychologie und der Ingenieurwissenschaften zielgerichtet zu einem gemeinsamen Vorgehen verknüpft werden, das über das Projekt hinaus Anwendbarkeit findet?

Dieser Beitrag stellt zunächst die Anforderungsanalyse als den Prozess heraus, in dem die aus unterschiedlicher Perspektive erhobenen Inhalte zusammen geführt werden müssen. Dazu werden zunächst Grundlagen aus beiden Bereichen zusammengetragen, um anschließend ein kooperatives Vorgehen zu definieren. Dieses Vorgehen wird im Projekt erprobt; erste Ergebnisse der Umsetzung werden erläutert und aus beiden Perspektiven diskutiert.

2 Grundlagen der szenariobasierten Anforderungsanalyse

Das beschriebene Modell der Interoperabilität stellt die semantische Ebene in den Mittelpunkt. In ihr treffen die Ziele der psychologischen und der ingenieurwissenschaftlichen Anforderungsanalyse zusammen: Während eine Anforderungsanalyse im Sinne der Psychologie Anforderungen an den Menschen zur Erfüllung von Aufgaben adressiert, zielt eine ingenieurwissenschaftliche Anforderungsanalyse konstruktiv auf die Festlegung von Produkteigenschaften ab. Beiden gemein ist dabei, dass Anforderungen auf Zielen beruhen. Somit können Parallelen im jeweiligen methodischen Ansatz erkannt werden: Bezogen auf den Forschungsgegenstand des Projekts LAGE kann zunächst angenommen werden, dass die psychologische Perspektive Stärken im Kontakt mit Akteuren verspricht, während die ingenieurwissenschaftliche Sicht auf die Formulierung lösungsorientierter Anforderungen fokussiert ist. Szenarien bilden einen gemeinsamen Kern und eine Schnittstelle zur Kommunikation zwischen allen Stakeholdern.

2.1 Psychologische Perspektive

Die Begriffe der Anforderung und der Anforderungsanalyse werden in der Psychologie in vielfältiger Wiese verwendet, es gibt keine eindeutige, allgemein anerkannte Definition (vgl. [Pi08]; Überblick in [Ul05]). Der Begriff der Anforderung bezieht sich dabei aber stets auf eine Person und das, was sie leisten oder können muss, um einer Situation oder einer Aufgabe gerecht zu werden. Es geht also nicht um die Anforderungen, die Nutzer an ein technisches System stellen, sondern um die Anforderungen, die die Situation oder Arbeitsaufgabe an die Person stellen. Generell geht es bei der Anforderungsanalyse darum, Anforderungen an Personen zu ermitteln

(z.B. Kompetenzen oder Fertigkeiten) sowie die Rahmenbedingungen, für die Bewältigung der Aufgabe. Aus Perspektive der Handlungsregulationstheorie werden Arbeitsaufgaben als Handlungsanforderungen verstanden [Ha98], die Regulationsprozesse der Person in Gang setzen, z.B. Denken, Planen, Zielbildung etc.

Im Kontext des hier vorgestellten Projekts sind die Handlungsanforderungen an Personen relevant, die in Strukturen und Prozesse verschiedener, kooperierender Organisationen eingebunden sind. Ziel der Erhebung war es, zentrale Handlungsanforderungen zu erfassen, die sich den Akteuren in der interorganisationalen Kommunikation und Kooperation stellen und dabei auch die Anforderungen der Arbeitsorganisation (vgl. [Sp98]) für die Situation „große Schadenslage“ zu erfassen. Die Analyseeinheit (der Einteilung von [Du99, S.12ff] folgend) ist eine Anforderungssituation „große Schadenslage“, die als Szenario beschrieben wird (s.u.). Die gewählte Perspektive ist die bedingungsbezogene Analyse, also von den konkreten Personen unabhängig. Als Merkmalsbereiche werden auf Organisationsebene Denk- und Planungserfordernisse und auf Ebene der Individuen Kooperations- und Kommunikationserfordernisse erfasst. Eine Voraussetzung für interorganisationale Kooperation ist die Herstellung eines „common ground“, also eines geteilten Bezugsrahmens [Cl96] bei der Bewältigung von großen Schadenslagen. Es ist anzunehmen, dass Lagebilder diese Funktion übernehmen können. Als Resultat für das erfolgreiche Herstellen eines „common ground“ durch den interorganisationalen Austausch von Lagebildern sollte ein „shared understanding‘, ein sogenanntes „gemeinsames mentales Modell“, in Bezug auf handlungsleitende Konzepte gebildet werden [TS01]. Es ist zu untersuchen wie Lagebilder die Herstellung eines gemeinsamen mentalen Modells unterstützen können und welche System-Anforderungen sich daraus ergeben.

Die verwendeten Methoden dienen dem Ziel, die interorganisationalen Arbeitsprozesse und die Strukturen, in denen sie stattfinden, zu verstehen. Es wird ein qualitativer Ansatz gewählt, der für explorative Studien besonders geeignet ist. Als Methoden für die Analyse der Handlungsanforderungen werden Dokumentenanalysen, Beobachtungen von Übungen entlang von Leitfäden (vgl. [Ku09]) und leitfadengestütze, halbstandardisierte Experteninterviews (vgl. [LT09]) verwendet. Diese verwenden, der critical incident- Methode [Fl54] folgend, Szenarien um die kontextsensitiven Wissensbestandteile der Stakeholder zu aktivieren.

2.2 Ingenieurwissenschaftliche Perspektive

Die ingenieurwissenschaftliche Sicht ist zunächst von Grundlagen der „klassischen“ Produktentwicklung (PE) sowie der Konstruktionslehre geprägt. Deren Vorgehensmodelle unterliegen einem signifikanten Einfluss durch das Qualitätsmanagement, das sich über Normen und entsprechende Zertifizierung übergreifend etabliert hat. Bezüglich der Betrachtung von Anforderungen über alle Entwicklungsschritte stellen Methoden des Requirements Engineering (RE), das in dieser expliziten Form dem Software Engineering (SE) zuzuordnen ist wichtige Erweiterungen dar, die einerseits in der Anwendung etabliert, andererseits in unterschiedlicher Tiefe weiter Forschungsgegenstand sind.

2.2.1 Perspektive Produktentwicklung (PE) und Qualitätsmanagement (QM)

Kundenanforderungen sind in der klassischen Produktentwicklung etabliert, um eine erfolgreiche Produktentwicklung zu gewährleisten. Sie stehen am Anfang jeder Entwicklungsaufgabe und entsprechend in Rahmenwerken zur Konstruktionsmethodik eingebettet (vgl. [PBFG07], [VDI2221]). Eine Anforderung wird nach DIN EN ISO 9000 [ISO9000] als „ein Erfordernis oder eine Erwartung, das oder die festgelegt, üblicherweise vorausgesetzt oder verpflichtend ist“ verstanden. Die klassische PE ist stark aufgabenorientiert ausgerichtet. Basis für das Erstellen einer Anforderungsliste ist das „klären und präzisieren der Aufgabenstellung“ [VDI2221], wobei die Anforderungsliste als „geklärte Aufgabenstellung für den Konstrukteur für den Konstruktionsbereich“ verstanden wird [PBFG07]. Dieser Schritt kann als grundlegende Festlegung einer Produktidee verstanden werden. In diesem Schritt erfolgt bereits eine Umfeldbetrachtung die weit über Produkt und Unternehmen hinausgeht, und sich mit Markt sowie MarktuUmfeld-Analysen beschäftigt. Das Ergebnis dieser Phase ist eine festgelegte Anforderungsliste, welche die Grundlage zur Freigabe der weiteren Entwicklungstätigkeiten bildet.

Diese Anforderungen werden im weiteren Verlauf nach der DIN 2519 [DIN2519] in Lasten- und Pflichtenhefte überführt. Bei genauerer Betrachtung des Produktentwicklungsprozesses wird klar, dass die Anforderungsanalyse sich als entwicklungsbegleitende Phase über den gesamten Konstruktionsprozess erstreckt. Pahl und Beitz betiteln dies als „anpassen der Anforderungsliste“. Anforderungen werden nach Kano [BBB93] in Bezug auf die Kundenzufriedenheit klassifiziert. Unterschieden wird in Basis-, Leistungs- und Begeisterungsmerkmale. In Bezug auf die Ermittlung von konkreten Kundenanforderungen bieten sich insbesondere die „Voice of the customer“-Methode und auch das Durchführen von „Lead User Workshops“ (vgl. [vHi86], [GRA06]) an. Die „Voice of Customer“-Methode ([BH97]) dient der Ermittlung der relevanten Kundenanforderungen durch ein Projektteam. Sie baut nicht auf einen strukturierten Fragebogen auf, sondern auf offenen Fragen in einem Interview [Be08]. Die „Lead User Methode“ zielt auf eine bessere Integration von Kunden in den Anforderungsprozess ab. Die Besonderheit hierbei ist, dass Kunden ausgewählt werden, die Innovationen stärker nutzen als der Rest einer Branche. Diese werden dann aktiv in den Entwicklungsprozess eingebunden.

Oftmals verschwimmen aber die Grenzen zwischen konkreter, auf ein Produkt bezogener Anforderungsanalyse und der strategischen Produktplanung innerhalb eines Unternehmens. Beide sind Teil der generellen Produktplanung (vgl. VDI 2221). Dies verdeutlicht das Beispiel der Anwendung von Szenarien: Während diese im Requirements Engineering stark auf die konkrete Anforderungsgenerierung ausgelegt ist (vgl. [Ru09]), finden sie in der klassischen Produktentwicklung eher als strategisches Tool zur langfristigen Entwicklung von Produktstrategien Anwendung (vgl. [GPW09]). Ein Szenario wird nach Gausemeier et al. definiert als „eine allgemeinverständliche Beschreibung einer möglichen Situation in der Zukunft, die auf einem komplexen Netz von Einflussfaktoren beruht ...“. Eine wichtige Erkenntnis dieses Konzeptes ist aber die Forderung nach der Erstellung mehrerer Szenarien, um möglichst viele Einflussfaktoren in Kalkül ziehen zu können [GPW09].

2.2.2 Perspektive des Software Engineering (SE)

Alle bedeutenden Vorgehensmodelle des Software Engineering beinhalten die Anforderungsanalyse als wesentlichen Schritt des Entwicklungsprozesses. Ihre Bedeutung wird in der Teildisziplin des „Requirements Engineering“ zusätzlich hervorgehoben. Dabei werden anforderungsbezogene Aktivitäten zu einem Rahmenwerk verknüpft, dass den gesamten Softwareentwicklungsprozess begleitet. Die Anforderungsanalyse oder -definition (vgl. [PR10], [Do97]) wird dabei als der Prozess verstanden, dessen Ergebnisse in einer Anforderungsspezifikation (vgl. Software Requirements Specification nach [IEEE830] bzw. im Sinne von Lasten- und Pflichtenheft nach [DIN69905] und [VDI2519]) dokumentiert werden. Diese nutzt zur Beschreibung sowohl Diagramme als auch natürlichsprachliche Formen. Letztere können durch schablonenbasierte Dokumentation unterstützt werden (vgl. besonders das Volere Schema nach [RR99] und die Anforderungsschablone nach [Ru09]).

Erster Schritt in der Anforderungsanalyse muss die Betrachtung und Abgrenzung von System, Kontext und Umgebung sein [Po08]. Besonders wichtig ist dabei die Analyse der Stakeholder. Dies sind „alle Personen, die von der Systementwicklung und natürlich auch vom Einsatz und Betrieb des Systems / Produkts betroffen sind.“ [Ru09]. Sie stehen insbesondere bei Anwendung benutzerzentrierter oder szenariobasierter Methoden im Fokus der Betrachtung (vgl. u. a. den „scenario based engineering process“ nach [MH97] und der Use Case-basierte Ansatz von [Ru94]). Szenarien werden dabei genutzt, um Ziele von Stakeholdern und damit verbundene Anwendungsfälle eines zu erstellenden Systems zu identifizieren. Dies ermöglicht die Ableitung lösungsorientierter Anforderungen (vgl. [Po08]). Die Variation einzelner Elemente eines Szenarios ermöglicht dabei einerseits die Fokussierung unterschiedlicher möglicher Nutzungskontexte, anderseits die Verifizierung von Anforderungen unter veränderter Ausgangslage. Insbesondere die Definition von Ereignissen kann zur Variantenbildung beitragen (vgl. „event scenarios“ nach [KS96]).

3 Methoden und interdisziplinäres Vorgehen

Auf Basis der erläuterten Grundlagen wurde für die Anforderungsanalyse im Projekt LAGE ein Konzept für die Durchführung einer kontinuierlichen, interdisziplinären und szenariobasierten Anforderungsanalyse (siehe Abbildung 1) erarbeitet. Als erster Schritt wird analog zur Phase „Planen und Klären der Aufgabenstellung“ aus dem Vorgehensmodell der PE eine intensive Recherche zu Strukturen und Prozessen der beteiligten BOS durchgeführt. In der SE wird besonders hervorgehoben, dass der erste Schritt einer Anforderungsanalyse die Betrachtung des Kontexts und der Umgebung sein sollte [Po08]. In besonderer Weise sollen hierbei die betroffenen Stakeholder berücksichtigt werden [Ru09]. Das zentrale Ziel ist zunächst, Erkenntnisse über die Zusammenarbeit der Organisationen im Falle einer großen Schadenslage zu erlangen. Die Literatur- und Dokumentenanalyse erfolgt in beiden Disziplinen als paralleler Prozess, um eine gemeinsame Ausgangsbasis zu schaffen. Im Vordergrund der Analyse steht die Untersuchung der organisatorischen Interoperabilität,um gezielt Anforderungen



Abbildung 1 Vorgehen bei der Interdisziplinären Anforderungsanalyse

für einen Meldungsaustauschstandard zu generieren. Methodisch soll hierfür ein szenariobasierter Ansatz als Grundlage dienen. Dieser Schritt bietet sich insbesondere aufgrund der hohen Komplexität großer Schadenslagen und der Vielzahl beteiligter Organisationen an. Da große Schadenslagen äußerst komplexe Situationen darstellen, werden dabei auch Aspekte basierend auf der Szenariotechnik der PE für die Anforderungsanalyse übernommen.

Die Basis für ein erstes ausformuliertes Szenario „Großschadenslage am Hauptbahnhof einer deutschen Großstadt" bildet die Literaturrecherche. Ziel ist somit ein skalierbares Szenario, welches durch verschiedene Varianten zur Klärung unterschiedlicher Fragen und zur Erhebung von Anforderungen dienen soll. Des Weiteren ermöglicht die flexible modulare Szenariengestaltung eine spätere Adaption auch auf andere Anwendungsfälle. Wie in Abbildung 1 zu erkennen ist, soll auch die Szenarienerstellung genau wie die Anforderungsanalyse und Dokumentation als iterativer Prozess über die gesamte Dauer des Projektes fortgeführt werden.

Anhand des ersten ausformulierten Szenarios werden Stakeholder identifiziert, die bei der Bewältigung des Szenarios zum Einsatz kommen würden. In diesem Teil der Anforderungsanalyse werden die Stakeholder Feuerwehr, Landes- und Bundespolizei, Technisches Hilfswerk, Deutsche Bahn AG, Deutsches Rotes Kreuz als Vertreter des Rettungsdiensts sowie den Krisenstab einer deutschen Großstadt als relevant festgelegt.

Aus psychologischer Sicht wird ein halbstandardisierter Interviewleitfaden für ein theoriengenerierendes Experteninterview entwickelt (vgl. [Fl07], [MN05]). Für die Entwicklung und Konstruktion des Interviewleitfadens wird zunächst eine qualitative Dokumentenanalyse auf der Basis von organisationsspezifischen Vorschriften, Webseiten und im Internet freizugänglichen Informationen der Stakeholder durchgeführt (vgl. [Ma02]). Für jeden Stakeholder werden die relevanten Dokumente in Bezug auf den Aufbau der Organisation, Führungsstrukturen, sowie interorganisationale Zusammenarbeit der BOS und Kommunikation interpretiert und kategorisiert. Aus den gebildeten Kategorien werden Themenblöcke zu interorganisationalen Prozessen und

Abläufen für den Interviewleitfaden formuliert. Zusätzlich gehen organisationsspezifische Fragen zu Aufbau- und Ablauforganisation für jeden Stakeholder in den Interviewleitfaden ein, um weiteres Betriebswissen zu erfassen (vgl. [MN05]). Bezüglich technischer Anwendungsprozesse werden verschiedene Abschnitte durch Fragen aus Ingenieursperspektive in Bezug auf den Einsatz von IT-Systemen bei Einsatz- und Führungsaufgaben sowie bei dem Austausch von Meldungen erweitert. Aus der SE-Sicht werden Interviews als Hilfsmittel zur Anforderungserhebung erwähnt. Analog hierzu ist das Voice of the Customer (VOC) Verfahren, welches seinen Ursprung im Qualitätsmanagement (QM, vgl. [TK09]) hat, erwähnt. Aus ingenieurwissenschaftlicher Sicht kommt somit eine Mischung aus VOC Methodik und halbstandardisierten Interviewleitfaden zum Einsatz.

Innerhalb des Interviews werden von beiden Disziplinen unterschiedliche Schwerpunkte verfolgt. Nach der Erstellung eines ersten Szenarios aus Ingenieurperspektive und der Erstellung des halbstandardisierten Interviewleitfadens aus psychologischer Sichtweise werden erste Experteninterviews mit Führungskräften der relevanten Stakeholder (vgl. „Lead User“) geführt. Sowohl Psychologen als auch Ingenieure führen gemeinsam die Interviews durch. Es erfolgt eine systematische Auswertung der Experteninterviews von beiden Disziplinen [Ma02]. Die Ergebnisse der Interviews werden von beiden Disziplinen systematisch ausgewertet und dienen als Basis zur Vorbereitung und Durchführung eines Workshops unter Beteiligung aller Stakeholder. Die Methodik dieses Workshops ist analog zur Lead User Methodik zu sehen, wobei die betrachteten BOS bereits eng mit Forschungseinrichtungen kooperieren und in ihrem Umfeld eine Lead User Position einnehmen.

4 Fallbeispiel: Ergebnisse im Projekt LAGE

Das Ergebnis dieser Vorgehensweise ist nach der Auswertung ein gemeinsam definiertes Szenario sowie ein solides Fundament von Anforderungen. Im Folgenden werden die fachlichen Ergebnisse dargestellt und anhand derer Erfahrungen bei der Durchführung der erarbeiteten Vorgehensweise erörtert. Die beschriebenen Ergebnisse stellen „Work-in-Progress“ dar, da die Anforderungsanalyse wie auch die weitere Verfeinerung der Szenarien projektbegleitend weitergeführt werden.

4.1 Szenariodefinition

Als initiales Szenario wurde zunächst ein Ereignis gewählt, das im Kreis der Stakeholder bekannt ist und die Notwendigkeit zur organisationsübergreifenden Kommunikation direkt impliziert. Am Beispiel einer deutschen Großstadt wurde eine Großveranstaltung mit Bezug zu einer kritischen Infrastruktur (Hauptbahnhof) herangezogen. Neben verschiedenen möglichen Schadensursachen bietet das Szenario zahlreiche alternative Einsatzverläufe, die im Projekt analysiert und einbezogen werden können. Wichtigste Parameter sind die „fremde Lage“, die sich aus Ereignis (Großveranstaltung, zusätzlich z. B. Anschlag), Ort (z. B. Unfall im Bahnhofsbereich) und Zeit (Auswirkung auf Randbedingungen) ergibt, sowie die „eigene Lage“ und damit insbesondere Einsatzkräfte und –mittel (vgl. [FWDV100], [Sc98]). Das Hauptziel der Gefahrenabwehr in diesem Szenario ist der Schutz von Menschen. Das Szenario

ist – abhängig von den gewählten Einflussgrößen – außerdem durch eine hohe räumliche Komplexität und eine große Dynamik der Lage sowie eine lange Dauer gekennzeichnet. Zur Unterstützung der Analyseschritte des Vorgehens wurde das Szenario modular gestaltet. Aus der Ingenieurperspektive deckt das Szenario damit die wichtigsten Anforderungen ab: Zunächst bildet es für das Fallbeispiel alle Anwendungsfälle aus der jeweiligen Stakeholdersicht ab. Diese lassen sich auf Grund der Festlegung auf Themen, die aktuell in Praxis und Forschung diskutiert werden (Großveranstaltungen, kritische Infrastrukturen, etc.), in der Domäne übertragen. Darüber hinaus ermöglicht das Szenario die Variation von Parametern für eine Sensitivitätsanalyse, um einerseits in einzelnen Anwendungsfällen Detailabweichungen zu identifizieren und andererseits Ergebnisse zu verifizieren. Außerdem bezieht es alle Stakeholder ein, die Einfluss auf die Gestaltung einer gemeinsamen Wissensbasis („gemeinsames Lagebild“) und die zu Grunde liegende Begriffswelt haben. Dies kann sukzessive erfolgen, so dass ein iteratives Vorgehen im Sinne aktueller Vorgehensmodelle (hier: Prozessanalyse und Ontology Engineering) möglich ist. Die kooperative Durchführung von Stakeholder-Befragungen sowie die gemeinsame Reflexion des Szenarios sowie der daraus abgeleiteten Anforderungen können dabei in allen Schritten der Anforderungsanalyse zur Validierung von Anwendungsfällen und Anforderungen an den Standardvorschlag „xHelp“ genutzt werden.

4.2 Analyse der Handlungsanforderungen

Aus der psychologischen Perspektive ergeben sich aus der Analyse des Szenarios unter anderem folgende Handlungsanforderungen:

- Es zeigen sich Unterschiede in den Organisations- und Führungsstrukturen verschiedener BOS („fehlende organisationale Passung“). Entscheidungs-relevante interorganisationale Schnittstellen müssen identifiziert werden.

- Das Informationsmanagement erfolgt auf verschiedenen Führungsebenen der Organisationen. Jede Führungskraft erfasst Informationen zur „großen Schadenslage“ und generiert aufgrund sog. Prüfpunkte ein organisationsspezifisches Lagebild. Relevante Informationen als Basis für Entscheidungen und Handlungen müssen identifiziert und kategorisiert werden. Informationen werden mit zunehmender Führungsebene verdichtet.
- Interorganisationale Kommunikation erfolgt auf den höheren Führungsebenen aufgrund des Austausches von Lagebildern, durch Fachberater und der Kommunikation von Führungskräften. Für den Austausch von Informationen besteht kein Kommunikationsstandard.
- Lagebilder werden nur in einigen BOS formalisiert erstellt. Der interorganisationalen Austausch von Lagebildern zwischen verschiedenen BOS erfolgt zur Dokumentation und ermöglicht wechselseitige Übersichten zur aktuellen Lage („gemeinsame mentale Modelle“).Beim Austausch zeigen sich neben der technischen Inkompatibilität auch Probleme in Bezug auf Semantik, Darstellungsform, Interpretation/Verständnis und Nutzung der Lagebilder (Lagebild zur: „Unterrichtung anderer BOS“ oder „Entscheidungsunterstützung“).

- Angesichts der Probleme beim Austausch von Lagebildern und zur Verbesserung der interorganisationalen Zusammenarbeit werden in Führungsstäben Fachberater bzw. Verbindungsbeamte eingesetzt. Fachberater repräsentieren aufgrund organisationsspezifischen Fachwissen und Einsatzerfahrung eine der wichtigsten „menschlichen" Schnittstellen der interorganisationalen Zusammenarbeit.

Die Analyse der Handlungsanforderungen verdeutlicht die Komplexität und Realitätsbezogenheit des gewählten Szenarios, insbesondere in Bezug auf die Auswirkungen der außergewöhnlichen sozialen Strukturen und das Ausmaß an psychischen Anforderungen.

4.3 Analyse der produktbezogenen Anforderungen

Aus der ingenieurwissenschaftlichen Perspektive ergeben sich folgende übergeordnete Anforderungen:

- Zum heutigen Zeitpunkt basiert die Kommunikation zwischen den verschiedenen Organisationen, sofern vorhanden, auf papiergebundenen Medien (Meldevordrucken), und auf der Arbeit der Verbindungsbeamten. Diese werde auch in Zukunft eine Rückfallebene zur Absicherung des Einsatzes von IT-gebundenen Lösungen darstellen. Somit muss beachtet werden, dass papiergebundene Kommunikation unterstützt, aber nicht vollständig ersetzt werden kann.
- Ein zentrales Element in der Kommunikation zwischen den Organisationen stellt die Arbeit der Verbindungsbeamten dar. Diese ist unabdingbar und kann nicht ersetzt werden. Hieraus ergibt sich die Anforderung, diese im Standard zu berücksichtigen.
- In den verschiedenen Organisationen herrschen unterschiedliche semantische Interpretationen von Begriffen und Symbolen. Die Verbindungsbeamten agieren hier oft als Mediator. Somit besteht die Anforderung diesen semantischen Diskrepanzen in einem Meldungsstandard entgegenzuwirken.
- Bestehende Meldevordrucke müssen bei der Entwicklung des Standards verstärkt Berücksichtigung finden und dienen als Basis für ein gemeinsames semantisches Verständnis.
- Die unterschiedlichen Aufbau- und Ablauforganisationen, wie sie in den verschiedenen Modulen des Szenarios diskutiert wurden, haben die Notwendigkeit aufgezeigt diese verstärkt zu berücksichtigen. Für den Standard tragen hier maßgeblich Prozessmodelle zur Abbildung dieser Strukturen bei.
- Insbesondere die Feuerwehr, welche in den meisten Szenarien die Einsatzleitung übernimmt, fungiert als Informationsdrehkreuz zwischen den Organisationen. Somit nehmen die Stäbe der FW eine zentrale Rolle für die Erstellung des Standards ein.
- Die Durchlaufzeit bei der Nachrichtenübermittlung ist laut der Interviews mit den Stakeholdern oft unbefriedigend. Der Prozess garantiert allerdings eine sichere Nachweisung. Somit muss für eine Akzeptanz des Standards diese Sicherheit gewährleistet, und die Dauer der Nachrichtenübermittlung verringert werden.

Bei genauerer Betrachtung kann festgestellt werden dass insbesondere spezifische Rollen und Prozessen in den Organisationen implizit Anforderungen an einen Standard stellen. Diese wurden durch das gewählte Vorgehen greifbarer gemacht. Der Standardvorschlag (Arbeitstitel „xHelp“) besteht, basierend auf den Resultaten der Anforderungsanalyse, aus drei Elementen: Das „xHelp-Format“ definiert das Datenformat des Standardvorschlags. Das „xHelp-Prozessmodell“ beinhaltet eine Abbildung aller bezüglich des Meldungsaustauschs relevanten Prozesse in Form von ereignisgesteuerten Prozessketten (vgl. [Sc01]). Eine „xHelp-Ontologie“ modelliert die bestehende Terminologie, setzt diese in Beziehung und identifiziert und spezifiziert übergreifende Begriffe (vgl. [SEM+01] auf Basis von [Gr93]).

5 Diskussion und Ausblick

Die Ergebnisse der bisherigen Forschung zeigen, dass das erarbeitete Vorgehen eine zielführende und mehrwerterzeugende Kooperation ermöglicht. Der szenariobasierte Ansatz ist geeignet, als Kommunikationsmedium psychologische und ingenieurwissenschaftliche Perspektiven zu verbinden. Im Rahmen der gemeinsamen Forschung, aber auch in der Kommunikation mit Akteuren kann ein Szenario zur Explizierung von Sachverhalten beitragen und damit zur Schaffung eines gemeinsamen mentalen Modells dienen. Das vorgeschlagene Vorgehen stellt Szenarien in den Vordergrund eines Vorgehens, das etablierte Vorgehensmodelle interdisziplinär integriert und eine inhaltliche sowie methodische Adaption ermöglicht. Das modular gestaltete Szenario basiert auf Grundlagen der betrachteten Domäne (System, Kontext und Umgebung) sowie Details aus einem konkreten Bedarfsfall. Zusätzlich ermöglicht es die Adaption an verschiedene Örtlichkeiten und Gegebenheiten um die Ergebnisse auf andere Anwendungsfälle zu übertragen. Es wird von Beginn an aus beiden Perspektiven erarbeitet und bildet damit alle relevanten Aspekte ab. Dies beinhaltet bereits wichtige Konsensbildung, die wechselseitig im jeweiligen Forschungsschwerpunkt genutzt werden kann. Das mehrstufige Vorgehensmodell wurde erprobt und kann nach Auswertung erster Ergebnisse auf Projekte mit vergleichbarer Aufgabenstellung bezüglich unterschiedlicher Ebenen der Interoperabilität übertragen werden. Insbesondere die zeitliche Dimension stellt dabei einen Faktor dar, bezüglich dessen der Nutzwert von Ergebnissen geprüft werden muss. Insbesondere für Forschungsprojekte kann konstatiert werden, dass ein hoher Aufwand für eine solche Analyse betrieben werden muss, der nicht unmittelbar auf eine technologische Lösung zielt, jedoch zur ganzheitlichen Betrachtung der Problemsituation notwendig ist. Im Projekt LAGE wurden erfolgreich verschiedenen Methoden der Anforderungsanalyse aus den ingenieurwissenschaftlichen und psychologischen Disziplinen miteinander verbunden. Hierbei hat insbesondere ein flexibler szenariobasierter Ansatz die Grundlage für die erfolgreiche Anforderungsermittlung gelegt. Die zusätzlichen psychologischen Erkenntnisse haben hierbei deutlich zu einem gesteigerten Systemverständnis, und somit auch zu einer erfolgreichen Anforderungsaufnahme beigetragen. Der in Abschnitt drei

beschriebene iterative Prozess wird noch bis Ende des Projektes fortgeführt. Die Kombination der verschiedenen Disziplinen in der Anforderungsanalyse hat sich als vorteilhaft herausgestellt. Nachteilig kann der hohe personelle und zeitliche Aufwand der Methode ins Feld geführt werden. Nach Abschluss der Evaluation des Standards „xHelp“ und somit auch des Projektes wird eine genaue Analyse des Anforderungsprozesses erfolgen. Ziel ist die Weiterentwicklung der interdisziplinären Methoden. Insgesamt gilt es zu prüfen wie der Ansatz weiter verbessert werden kann. Der Fokus hierbei sollte auf einer Verringerung des Durchführgsaufwands liegen.

Literaturverzeichnis

[BBB93] Berger, C.; Boger, D.; Bolster, C.: Kano’s Methods of understanding customer defined quality; 1993 The Journal of Japanese Society for Quality Control 1993

[Be08] Becker, T.: Prozesse in Produktion und Supply Chain optimieren, 2008 Springer Verlag Berlin Heidelberg

[Cl96] Clark, H.H: Using language. Cambridge: Cambridge University Press, 1996.

[DIN69905] DIN 69905: Projektwirtschaft; Projektabwicklung; Begriffe, 1997.

[Do97] Dorfman, M.: Requirements Engineering. Software Requirements Engineering. In (Thayer, R.; Dorfman, M. Hrsg.): Software requirements engineering. IEEE Computer Soc. Press, Los Alamitos, Calif., 1997; S. 7–22.

[Du99] Dunckel, H., Psychologische Arbeitsanalyse: Verfahrensüberblick und Auswahlkriterien. In H. Dunckel (Hrsg.), Handbuch psychologischer Testverfahren., Zürich: vdf, 1999, S. 9-30.

[Fa98] Faulk, S. R.: Software Requirements: A Tutorial, 1998.

[Fl54] Flanagan, J.C.: The Critical Incident Technique. Psychological Bulletin. 51, 4, 1954,

[Fl07] Flick, U.: Qualitative Sozialforschung – Eine Einführung. Reinbek: Rowohlt, 2007.

[FWDV100] Feuerwehr-Dienstvorschrift 100: Führung und Leitung im Einsatz. Führungssystem. Deutscher Gemeindeverlag, Stuttgart, 2003.

[GPW09] Gausemeier, Jürgen; Plass, Christoph; Wenzelmann, Christoph (2009): Zukunftsorientierte Unternehmensgestaltung. Strategien, Geschäftsprozesse und IT-Systeme für die Produktion von morgen.

[Gr93] Gruber, T. R.: A Translation Approach to Portable Ontology Specifications. In: Knowledge Acquisition Vol. 6 (1993), Nr. 2, 1993, S. 199–221

[Ha98] Hacker, W., Allgemeine Arbeitspsychologie. Bern: Huber, 1998.

[vHi86] von Hippel, E. (1986), "Lead Users: A Source of Novel Product Concepts", Management Science 32(7): 791–806

[IEEE610] IEEE Std 610-1990 - IEEE Standard glossary of software engineering terminology, IEEE Press, Piscataway, New Jersey, 1990.

[IEEE830] IEEE Std 830-1998 - IEEE Recommended Practice for Software Requirements Specifications, IEEE Press, Piscataway, New Jersey, 1998.

[ISO9000] ISO 9000: Qualitätsmanagementsysteme - Grundlagen und Begriffe

[KS96] Kotonya, G.; Sommerville, I.: Requirements engineering with viewpoints. In Software Engineering Journal, 1996, 11; S. 5–18.

[Ku09] Kuhlmann, M., Beobachtungsinterview. In S. Kühl, P. Strodtholz, & Taffertshofer, A. (Hg.). Handbuch Methoden der Organisationsforschung. Wiesbaden: Verlag für Sozialwissenschaften, 2009, S. 78-102.

[LPKH10] Lindemann, C., Pottebaum, J., Koch, R., Held, C.: Standardisation of Reports to Optimise Cooperation in the Domain of Public Safety and Security. In: Proceedings

of Future Security 2010, 2010
[LT09] Liebold, R., Trinczek, R.: Experteninterview. In S. Kühl, P. Strodtholz, & Taffertshofer, A. (Hg.). Handbuch Methoden der Organisationsforschung. Wiesbaden: Verlag für Sozialwissenschaften, 2009, S. 32-57.
[Ma02] Mayring, P.: Einführung in die Qualitative Sozialforschung – Eine Einleitung zu qualitativem Denken (5.Auflage). Weinheim/Basel: Beltz, 2002.
[MH97] McGraw, K.; Harbison, K.: User-centered requirements: the scenario-based engineering process. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1997.
[MN05] Meuser, M., Nagel, U.:.ExpertInneninterviews – vielfach erprobt, wenig bedacht. Ein Beitrag zur qualitativen Methodendiskussion. In: Bogner, A., Littig, B. & Menz, W. (Hrsg.). Das Experteninterview – Theorie, Methode, Anwendung (2., Aufl.). Wiesbaden: VS Verlag für Sozialwissenschaften, 2005, S.71-93.
[PBFG07] Pahl, Gerhard; Beitz, Wolfgang; Feldhusen, Jörg; Grote, Karl-Heinrich; Pahl-Beitz: Konstruktionslehre. Grundlagen erfolgreicher Produktentwicklung ; Methoden und Anwendung. 7. Aufl. Berlin: Springer (Springer-Lehrbuch).2007
[Pi08] Pixner, J (2008). Erfolgskritische Anforderungen im Hochschulstudium. Dissertation an der Albert-Ludwigsburg-Universität Freiburg. Online verfügbar unter http://www.freidok.uni-freiburg.de/volltexte/5939/pdf/Diss_Pixner_Final_.pdf
[PJPK10] Pottebaum, J., Japs, A. M., Prödel, A., Koch, R.: Design and modelling of a domain ontology for fire protection. In: French, Simon; Tomaszewski, B.; Zobel C. (Hg.): Proceedings of the 7th International Conference on Information Systems for Crisis Response and Management ISCRAM2010, 2010
[Po08] Pohl, K.: Requirements engineering. Grundlagen, Prinzipien, Techniken. dpunkt-Verl., Heidelberg, 2008.
[PR10] Pohl, Klaus; Rupp, Chris : Basiswissen Requirements Engineering. Aus- und Weiterbildung zum "Certified Professional for Requirements Engineering" ; Foundation Level nach IREB-Standard. 2., aktualisierte Aufl. Heidelberg: dpunkt-Verl.2010
[RR99] Robertson, S.; Robertson, J.: Mastering the requirements process. Addison-Wesley, Harlow, 1999.
[Ru94] Rumbaugh, J. E.: Getting Started: Using Use Cases to Capture Requirements. In Journal of Object-Oriented Programming, 1994, 7; S. 8-12, 23.
[Ru09] Rupp, Chris : Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis. 5., Aufl. München: Hanser.2009
[Sc98] Schläfer, H.: Das Taktikschema. Kohlhammer, Stuttgart, 1998.
[Sc01] Scheer, A.-W.: ARIS - Modellierungsmethoden, Metamodelle, Anwendungen. Springer, Berlin, 2001.
[SEM+01] Studer, R., Erdmann, M., Mädche, A. D., Oppermann, H., Schnurr, H.-P., Staab, S., Sure, Y., Tempich, C.:Arbeitsgerechte Bereitstellung von Wissen:Ontologien für das Wissensmanagement;. Arbeitsbericht des Instituts AIFB, Universität Karlsruhe, 2001
[Sp98] Spur, G.,. Automatisierung und Wandel der betrieblichen Arbeitswelt. Berlin: De Gruyter, 1998.
[TK09] Toutenburg, H.; Knöfel, P.: Six Sigma Methoden und Statistik für die Praxis; Springer Verlag Heidelberg, 2009
[TS01] Tschan, F., Semmer, N.: Geteilte mentale Modelle und Leistung in der Teamarbeit. In: Fisch, R., Beck, D. & Englich, B. (Hg.): Projektgruppen in Organisationen. Göttingen:Hogrefe. 2001; S.217-235.
[Ul05] Ulich, E (Hrsg.), Arbeitspsychologie, 6. Auflage. Zürich: vdf/Schäffer-Pöschel, 2005
[VDI2221] VDI-Richtlinie 2221 : Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte, 1993
[VDI2519] VDI-Richtlinie 2519 - Vorgehensweise bei der Erstellung von Lasten-/Pflichtenheften, 2001.

Requirements Engineering: Ein Ansatz auch für die klassische Produktentwicklung?

Christian Lindemann, Kurt-Robert Hippler, Rainer Koch

Universität Paderborn
Computeranwendung und Integration in Konstruktion und Planung (C.I.K.)
Pohlweg 47-49
33098 Paderborn
{lindemann;r.koch@cik.upb.de}
krh@mail.uni-paderborn.de

Abstract: Eine systematische Analyse von Kundenanforderungen wird in der heutigen Zeit immer wichtiger. In den Disziplinen der klassischen Produktentwicklung (PE) und des Software Engineerings (SE) gibt es hierzu vielfältige Systematiken und Tools welche sich mit dieser Analyse dieser Anforderungen beschäftigen. Diese sind in der Regel in etablierte Methoden zur Produktentwicklung eingebettet. In dieser Arbeit wird ein interdisziplinärer Ansatz zur Produktentwicklung in der elektromechanischen Industrie entwickelt und vorgestellt. Hierbei werden Synergieeffekte der unterschiedlichen Methoden und Ansätze genutzt um eine branchenspezifische Methodik zu entwickeln.

1 Einleitung

In der heutigen Zeit stehen Unternehmen bei der Produkterstellung immer komplexeren Herausforderungen gegenüber. Produkte sollen kundenindividuell, innovativ und hochwertig sein. Durch verkürzte Produktlebenszyklen müssen diese Produkte in immer kürzeren Abständen zu einem immer günstigeren Preis auf den Markt gebracht werden[SBA02]. Aufgrund eines immer höheren Wettbewerbs auf den Märkten, insbesondere durch Nachahmer, erodieren erreichte Wettbewerbsvorteile immer schneller [Bu94]. Insbesondere die Summe dieser Herausforderungen verlangt ein systematisches, reproduzierbares Vorgehen bei der Produktentwicklung. Hierzu bieten das Software Engineering sowie die klassische Produktentwicklung in der Mechanik diverse Ansätze um eine erfolgreiche Verfahrensweise zu gewährleisten.

Um die Kundenforderungen bezüglich neuer Produkte möglichst genau zu erfüllen und Wettbewerbsvorteile aufrecht zu erhalten, bedarf es einer gezielten Aufnahme der Kundenwüsche. Hierzu bieten die verschiedenen Disziplinen in den Entwicklungsprozess eingebettete Methoden der Anforderungserhebung an. Insbesondere das Requirements Engineering (RE) als Teil-Disziplin des Software Engineering befasst sich mit dem Finden, Dokumentieren und Pflegen von Kundenanforderungen an ein (Software-)Produkt. Der Begriff Engineering sagt aus, dass es sich um einen systematischen Prozess handelt [PR10], der reproduzierbar möglichst alle relevanten Kundenanforderungen erhebt.

Ziel der Untersuchungen ist es die Anwendbarkeit von Techniken des Requirements Engineerings am Beispiel eines elektromechanischen Produktes im Hinblick auf die klassische Produktentwicklung zu untersuchen. Die Diskussion wird bezogen auf ein entwickeltes und bereits prototypisch in einem Unternehmen eingeführtes interdisziplinäres Verfahren zur Anforderungsanalyse. Es wird exemplarisch ein reales Fallbeispiel aus der Industrie diskutiert und analysiert.

2 Einordnung der Anforderungsanalyse in den Konstruktionsprozess

Um die Produktentwicklungsprozesse hinsichtlich der Vorgehensweisen bei der Anforderungsanalyse zu vergleichen, erfolgt an dieser Stelle eine kurze Betrachtung der Systematiken, in welche die Erhebung von Anforderungen eingebettet ist. Hierbei werden insbesondere die ingenieurwissenschaftlichen Systematiken aus dem Software Engineering und der klassischen Produktentwicklung betrachtet. An dieser Stelle wird vertretend für die beiden Disziplinen jeweils ein Vorgehensmodell vorgestellt.

2.1 Anforderungsanalyse in der klassischen Produktentwicklung (PE)

In der klassischen Produktentwicklung existieren viele verschieden Methoden, zur systematischen Produktentwicklung. Zu nennen sind hierbei insbesondere die „Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte“ nach der VDI 2221 [VDI2221], der Produktentstehungsprozess nach Pahl/Beitz [PBFG07] als auch das Algorithmische Auswahlverfahren zur Konstruktion mit Katalogen nach Roth [Ro00]. Hierbei stellt die Erhebung von Kundenanforderungen ein zentrales Element dar. Anforderungen werden nach der VDI 2221 als „*qualitative und/oder quantitative Festlegung von Eigenschaften oder Bedingungen für ein Produkt“* [VDI2221] oder nach ISO 9000 als „*ein Erfordernis oder eine Erwartung, das oder die festgelegt, üblicherweise vorausgesetzt oder verpflichtend ist*“ [ISO9000] definiert.

Repräsentativ soll an dieser Stelle nur das Vorgehen nach PAHL/BEITZ beschrieben werden. Es gliedert den Produktentstehungsprozess in vier Phasen[1] mit unterschiedlichen Aufgabenpaketen: [PBFG07] planen und klären der Aufgabe, konzipieren, entwerfen und ausarbeiten. Den Start der ersten Phase markiert entweder ein konkreter Kundenauftrag oder eine innovative Produktidee im Unternehmen selbst. Diese Produktidee wird auf Durchführbarkeit in Form eines Produktes geprüft. Dafür sind konkrete Produktvorschläge zu erarbeiten. Die daraus resultierende Entwicklungsaufgabe ist dann genauer zu beschreiben. Dies wird als Klärung der Aufgabe bezeichnet. Zur Klärung der Aufgabe ist in den meisten Fällen eine umfassende Informationsbeschaffung unternehmensinterner- und externer Daten notwendig.

Dabei werden die Anforderungen an das Produkt herausgearbeitet und in Form einer Anforderungsliste zusammengefasst. Den Abschluss dieser Phase bildet dann die Freigabe zum Konzipieren des neuen Produktes.

[1] Für die Thematik der Anforderungsanalyse sind nur die ersten zwei Phasen Interessant, daher wird auf eine Beschreibung der letzten Phasen verzichtet.

Konzipieren: In dieser zweiten Phase folgt die Entwicklung einer prinzipiellen Lösung für das Produkt. Dies geschieht durch das Abstrahieren der gesammelten Anforderungen. In dieser Phase werden Funktionsstrukturen aufgestellt, Teilfunktionen werden zu einer Gesamtfunktion zusammengefügt. Durch die Suche nach geeigneten Wirkprinzipien und das Zusammenfassen in einer Wirkstruktur[2] wird die prinzipielle Lösung für das Produkt festlegt. „Das Konzipieren ist die prinzipielle Festlegung einer Lösung." [PBFG07] Für die Darstellung der Lösung können unterschiedlichste Dokumente verwendet werden. Dabei sind teilweise Handskizzen ausreichend, um eine prinzipielle Lösung zu verdeutlichen. Die gesamte Phase wird hierbei in mehrere Arbeitsschritte unterteilt [PBFG07]. Die erarbeiteten Lösungsvarianten müssen auf die Erfüllung der Forderungen der Anforderungsliste überprüft werden. Im Falle der Nichterfüllung wird die Variante verworfen. Das Erarbeiten prinzipieller Lösungen muss äußerst gewissenhaft durchgeführt werden, da Mängel des Lösungsprinzips in der Phase des Entwerfens nur schwer bzw. gar nicht behoben werden können.

2.2 Anforderungsanalyse im Software Engineering (SE)

Im Software Engineering befasst sich insbesondere die Disziplin des Requirements Engineering mit der Erhebung der Anforderungen von Stakeholdern. Unter den Begriff RE fallen alle Aktivitäten, die mit der Entwicklung und Dokumentation von Anforderungen an ein (Software-)Produkt zu tun haben. Auch in dieser Systematik ist die Erhebung von Anforderungen ein Prozessschritt, der zu Beginn eines jeden Produktentwicklungszyklus steht. Anforderungen gelten als eine Bedingung oder Eigenschaft, die ein System benötigt, um ein Problem zu lösen oder ein Ziel zu erreichen [IEEE90]. Hierbei sind unter dem Begriff System[3] sowohl physische Produkte als auch Dienstleistungen, wie z. B. die Erstellung von Software, zu verstehen. Rupp definiert eine Anforderung als „*Aussage über eine Eigenschaft oder Leistung eines Produktes, eines Prozesses oder der am Prozess beteiligten Personen*" [Ru09]. Ausgehend von einer Betrachtung und Abgrenzung von System, Kontext und Umgebung rücken die Stakeholder[4] eines Systems in den Mittelpunkt der Anforderungsanalyse [Po08].

An dieser Stelle wird repräsentativ nur das Vorgehen von ROBERTSON/ROBERTSON beschrieben. Das Modell stellt einen allgemeingültigen dreistufigen Entwicklungsprozess vor, der sich aus dem Requirements Process, der Systemanalyse und dem Product Design zusammensetzt [RR06]. Die Autoren trennen den Prozess der Anforderungsanalyse und –spezifikation aus der Systemanalyse heraus und stellen ihn dem gesamten Entwicklungsprozess als Stufe des Requirements Process voran. In diesem Prozessschritt werden alle Anforderungen betreffenden Aktivitäten durchgeführt. Dies geschieht immer unter der Berücksichtigung der Stakeholder-Interessen und der geplanten Umgebungsbedingungen des Produkts. Als Ergebnis dieser Stufe liegen die Requirements Specifications vor, welche auch bei diesem Vorgehen in Form eines Anforderungskatalogs oder

[2] Ein Wirkprinzip ist ein Grundsatz, von dem sich eine bestimmte Wirkung zur Erfüllung der Funktion ableitet
[3] EHRLENSPIEL definiert ein System als eine Menge von Elementen, die bestimmte Eigenschaften aufweisen und durch unterschiedliche Beziehungen untereinander verbunden sind. Systeme grenzen sich durch die Systemgrenze von ihrem Umfeld ab. [Eh09]
[4] Dies sind „alle Personen, die von der Systementwicklung und natürlich auch vom Einsatz und Betrieb des Systems/Produkts betroffen sind" [Ru09].

einer Anforderungsliste gesammelt werden. Anschließend daran erfolgt die Systems Analysis, die dem Requirements Process sehr ähnlich ist und sich in einigen Arbeitsschritten auch mit dieser Stufe des Prozesses deckt bzw. sich teilweise überschneidet. Zum Abschluss dieser Stufe wird das Produkt komplett verstanden. [RR06] Es liegen sowohl die Requirements Specification als auch die Analysis Specification vor. Anschließend erfolgt das eigentliche Product Design. Hierbei werden die teilweise noch abstrakten Vorstellungen aus den Requirements und Analysis Specifications in ein endgültiges Design umgesetzt. Ergebnis dieser Stufe ist die Design Specification, die als Fertigungsunterlagen für die Herstellung des Produkts benötigt werden. Die Stufen Build und Product Usage stellen keine Stufen des eigentlichen Entwicklungsprozesses mehr dar. So wirken sich diese Stufen auf die Anforderungsanalyse aus, da ein Feedback aus diesen Stufen in den Requirements Process einfließt. Das Vorgehen ist iterativ und empfiehlt Rücksprünge in vorangegangene Stufen des Entwicklungsprozesses. Die Anforderungsanalyse und –spezifikation nimmt bei diesem Modell jedoch eine Sonderstellung ein, da für die Erhebung von Anforderungen und deren Dokumentation erstmals ein eigenständiger Prozessschritt eingeführt wurde.

2.3 Anforderungsanalyse im Konstruktionsprozess

Abbildung 1 stell noch einmal die bei der Analyse betrachteten Verfahren gegenüber, welche die Grundlage für die Entwicklung eines produktspezifischen Ansatzes zur Anforderungsanalyse darstellen.

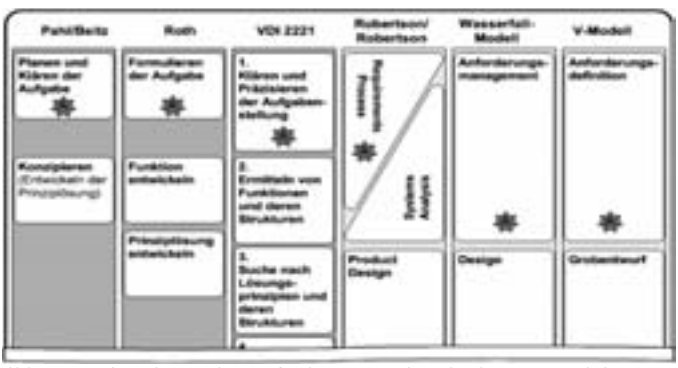


Abbildung 1: Einordnung der Anforderungsanalyse in den Konstruktionsprozess

Schon auf den ersten Blick wird deutlich, dass die Tätigkeit der Anforderungsanalyse und -spezifikation in allen Modellen zu einem sehr frühen Zeitpunkt des Entwicklungsprozesses erfolgt. Hierbei zeichnen sich alle Modelle durch die Möglichkeit der iterativen Produktentwicklung aus. Somit sind durchaus Sprünge zwischen den verschiedenen Entwicklungsphasen möglich. Als Ergebnis der ersten Phasen steht grundsätzlich das Dokument der Anforderungsliste. Dies kann als Anforderungsspezifikation (vgl. Software Requirements-Specification nach [IEEE830]) oder im Sinne der klassischen Produktentwicklung von Lasten- und Pflichtenheft nach [DIN69905] und [VDI2519] dokumentiert werden. Mit Ausnahme des Wasserfallmodels ermöglichen alle Modelle während der gesamten Produktentwicklung eine flexible iterative Anpassung

der Anforderungsdefinition und Dokumentation. Insbesondere die Modelle des Software Engineerings ermöglichen eine Änderung bis zur Produktfertigstellung. Bei genauer Betrachtung wird deutlich, dass die verschiedenen Verfahren durchaus Überschneidungen aufweisen. Dies gilt nicht nur für die klassischen Modelle der Produktentwicklung, sondern auch für die Modelle und Vorgehensweisen, die aus der Softwareerstellung stammen. Hierbei ist das Modell von Robertson/Robertson hervorzuheben, da sich dies explizit nicht nur auf Softwareprojekte übertragen lässt. Auffällig sind die Überschneidungen jedoch besonders bei den klassischen Vorgehensmodellen. So kommt Roth zu dem Schluss: „*Die Ablaufpläne von Roth und Pahl/Beitz stimmen mit dem Ablaufplan der Richtlinie VDI 2221 genau überein*." [Ro00]. Als grundsätzlicher Unterschied zwischen den Modellen der Mechanik und der Softwareentwicklung lässt sich die feinere Struktur der Mechanik-Modelle herausstellen. Die Modelle umfassen zwar teilweise ebenfalls, wie die Vorgehensweisen der Softwareentwicklung vier Phasen, sind dafür aber klarer in Form von Arbeitspaketen strukturiert. In einem Punkt stimmen jedoch alle Modelle überein: Die Anforderungsanalyse und –spezifikation erfolgt immer in den ersten Prozessschritten der jeweiligen Methode. Die erarbeitete Anforderungsliste wird bei fast jedem Modell, beim Durchlaufen der Methode, iterativ weiterentwickelt und von den nachfolgenden Prozessschritten weiter beeinflusst.

3 Interdisziplinäre Anforderungsanalyse in der Praxis

In diesem Abschnitt wird die Entwicklung einer produktspezifischen interdisziplinären Systematik zur Anforderungsanalyse beschrieben. Dabei wird die Methode exemplarisch anhand eines Unternehmens aus dem Segment der elektromechanischen Industrie untersucht. Hierbei wurde versucht die Synergieeffekte der Ansätze der klassischen Produktentwicklung und des Softwareengineering zu nutzen. Im vorigen Abschnitt wurde bereits erläutert, wie einzelne Richtlinien [VDI2221] [ISO9000] [IEEE90] oder Autoren [Ru09] den Anforderungsbegriff verstehen. Um ein gemeinsames Verständnis für den folgenden Abschnitt zu schaffen, wird an dieser Stelle eine eigene Definition eingeführt. *„Anforderungen ermöglichen qualitative und/oder quantitative Aussagen über Eigenschaften von Produkten bzw. Prozessen, mit teilweise unterschiedlichen Gewichtungen. Sie sind notwendig um Systeme zu beschreiben, Probleme zu lösen oder Ziele zu erreichen."*

Im Folgenden werden die Anforderungen an eine produktsegmentspezifische Methode zur Anforderungsanalyse und vorgestellt. Für das betrachtete Produkt sind die nachfolgend aufgelisteten zehn Forderungen mit dem betrachteten Unternehmen als signifikant herausgearbeitet worden.:

1. Einfache Generierung von Anforderungen und Anforderungsliste
2. Überschaubarer Aufwand für die Durchführung der Methode
3. Möglichst vollständige Ermittlung der Anforderungen
4. Überführbarkeit in ein Lasten- / Pflichtenheft
5. Anforderungsgenerierung aus Normung und Zulassung
6. Einbeziehung ausländischer Niederlassungen und Tochtergesellschaften
7. Verwendung von Markt- und Wettbewerbsdaten
8. Kosten und Preise als Kernanforderung

9. Einbindung von Kunden in die Anforderungsgenerierung
10. Konfliktmanagement bei widersprüchlichen Anforderungen

Grundsätzlich können diese Anforderungen an die Methode jedoch auch für andere Unternehmen oder Märkte interessant sein. Die Anforderungen an die Methode zielen insbesondere auf die Integration von Kundenwünschen, die Verwendung von Markt- und Wettbewerbsdaten und unternehmensinterner Ideen ab.

3.1 Bewertung der Methoden hinsichtlich der Anforderungserfüllung

Um die gestellten Anforderungen an einen neuen, produktspezifischen Prozess zur Anforderungserhebung zu erfüllen, wurde hauptsächlich auf die bereits in Abschnitt zwei beschriebenen Methoden zurückgegriffen. Zusätzlich wurden weitere erfolgversprechende Techniken[5] aus dem weiteren Umfeld der beiden Ansätze betrachtet (siehe Abbildung 2).

Die Vorgehensmodelle der Mechanik bieten gute Ansätze zur Anforderungsanalyse. Die Vorteile des Modells liegen dabei vor allem in der Vielzahl vorgeschlagener weiterführender Methoden zur Anforderungsanalyse (vgl.[VDI2221]). Diese Methoden sind jedoch nicht direkt in den Prozess integriert. Die Ergebnisse, die von diesem Modell geliefert werden, hängen somit stark von jeweiligen Bearbeiter und seinem Fach- und Methodenwissen ab. Vorteilhaft ist, dass oftmals die Grenzen zwischen konkreter, auf ein Produkt bezogener Anforderungsanalyse und der strategischen Produktplanung verschwimmen. Beide sind Teil der generellen Produktplanung (vgl. VDI 2220 bis VDI 2222). Dies verdeutlicht das Beispiel der Anwendung von Szenarien: Während diese im Requirements Engineering stark auf die konkrete Anforderungsgenerierung ausgelegt ist (vgl.[Ru09]), finden sie in der klassischen Produktentwicklung eher als strategisches Tool zur langfristigen Entwicklung von Produktstrategien Anwendung (vgl.[GPW10]).
Auch das Requirements Engineering weist hinsichtlich der gestellten Anforderung Schwachstellen auf. Oft basiert das Vorgehen auf der Vorstellung, für einen „bekannten“ Kunden zu produzieren. Im Markt für das elektromechanische Produkt wird für einen „anonymen“ Kundenkreis hergestellt, diese Produkte werden dann auf dem Markt einer Vielzahl von Kunden angeboten. Es ist also nicht garantiert, dass Kunden automatisch zur Gruppe der Stakeholder gehören, die sich aktiv am Entwicklungsprozess beteiligen. Die Vorstellung über die enge Zusammenarbeit mit dem Kunden als Stakeholder resultiert beim Requirements Engineering Ansatz aus seiner Herkunft, da bei der Softwareerstellung oftmals kundenspezifische Projekte durchgeführt werden.

[5] Die Techniken stehen nicht im Kern der Arbeit, und werden nur kurz angerissen. Weiterführende Literatur wird angegeben.

Fragestellung: Wie erfüllen die unten gelisteten Vorgehensmodelle und Methoden die an die Systematik gestellten Anforderung? ++ weitestgehend erfüllt + teilweise erfüllt 0 in Ansätzen erfüllt -- nicht erfüllt	Anforderungen an die Methode									
	1	2	3	4	5	6	7	8	9	10
	Einfache Generierung von Anforderungen und Anforderungslisten	Möglichst vollständige Erfassung der Anforderungen	Überschaubarer Aufwand für die Durchführung	Überführbarkeit in ein Lasten- / Pflichtenheft	Anforderungsgenerierung aus Normung und Zulassung	Einbeziehung ausländischer Niederlassungen und Tochtergesellschaften	Verwendung von Markt- und Wettbewerbsdaten	Kosten und Preise als Kennanforderung	Einbindung von Kunden in die Anforderungsgenerierung	Konfliktmanagement bei widersprüchlichen Anforderungen
Pahl/Beitz	+	+	+	++	+	0	+	--	0	0
Robertson/Robertson	+	+	+	++	0	+	0	--	++	++
Marktanalyse	0	0	+	+	--	++	++	+	0	--
Wettbewerbsanalyse	0	0	+	+	--	++	++	+	0	--
Lead-User-Methode	+	+	--	++	0	0	0	+	++	0
Conjoint-Measurement	+	+	--	+	0	+	0	++	++	+
QFD	0	0	0	+	+	0	+	0	++	+
Szenario-Technik	0	+	--	+	--	+	0	+	+	--

Abbildung 2: Gegenüberstellung Anforderungen und Vorgehensmodelle/Methoden[6]

Markt- und Wettbewerbsanalysen[7] dienen vor allem der Bereitstellung von Markt- und Wettbewerbsdaten. Der Aufwand für die Durchführung ist je nach gewählter Methode überschaubar, sofern die Daten für die Untersuchung verfügbar sind.Somit sollten diese Analysemethoden als Alternative, in der neuen Methode Berücksichtigung finden. Für die Einbeziehung der beiden Methoden spricht auch die teilweise Eignung zur Ermittlung von Preis-Anforderungen und technischen Wettbewerbsvergleichen.

Auf Grund ihrer guten Eignung für die Integration des Kunden in den Entwicklungsprozess sollte die Lead-User-Methode (vgl. [vHi86]) als ergänzende Methode zum Prozess der Anforderungsanalyse und – spezifikation genutzt werden. Der Gewinn, den neue innovative Anforderungen für die (Neu-) Produktentwicklung bedeuten, ist dabei nicht zu unterschätzen.

Obwohl die Methode Quality Function Deployment (QFD) (vgl. [No06]) zumindest eine Erfüllung in Ansätzen bietet und bei keiner der Anforderungen des Produktsegments „durchfällt“ sollte diese Methode trotzdem nicht in die neue Methode zur Anforderungs-

[6] Auf Grund der hohen Überschneidungsrate zwischen den Modellen der Mechanik wurde im Vergleich nur noch das Modell von PAHL/BEITZ stellvertretend für alle anderen betrachtet.
[7] Zur besseren Darstellung wurden die Markt- und Wettbewerbsanalyse zusammengefasst. Betrachtet wurden stellvertretende insbesondere Porters 5 Forces Modell[Po80], sowie das BCG Portfolio [KKB07].

analyse und –spezifikation einfließen. Den Grund hierfür stellt wieder der Aufwand für die Durchführung der Methode dar. Zwar lässt sich mit Blitz-QFD nach ZULTNER (vgl. [Zu95]) das Verfahren abkürzen, jedoch werden bei diesem Ansatz bestimmte Kundenkreise (kleinere Kunden) in der Untersuchung nicht betrachtet.

Im Rahmen der Anforderungsanalyse und –spezifikation kann die Szenario-Technik (vgl. [GPW10]) ihre strategischen Vorteile nicht ausspielen. Für die operative Erhebung und Dokumentation von Anforderungen im Entwicklungsprozess ist diese wenig geeignet. Bei langfristig angelegten Strategieprojekten könnte die Szenario-Analyse zur Ermittlung von Trends herangezogen werden. Aus diesen Trends ließen sich Rückschlüsse auf mögliche Produktanforderungen ziehen.

Grundsätzlich bieten sowohl das Vorgehensmodell der Mechanik als auch das Modell des Requirements Engineering gute Ansätze zur Durchführung der Anforderungsanalyse für das betrachtete Produktsegment. Jedes der betrachteten Modelle erfüllt Teile der Anforderungen an die Methode. Aufgrund der Betrachtung des Verhältnisses von Aufwand zu Ergebnissen fallen sowohl die Szenario-Technik als auch das QFD Technik aus dem Methodenpool zur Entwicklung heraus. Hierbei bleibt zu prüfen, ob diese Techniken an anderer Stelle Einsatz finden können.

3.2 Interdisziplinärer Prozess

Bei der Entwicklung einer neuen Methodik wird der aufgezeigte Handlungsbedarf unter Berücksichtigung der Ergebnisse aus 3.1 in eine neue bzw. modifizierte Methode umgesetzt. Nach Rupp müssen hierzu die Fragen nach dem Vorgehensmodell, den Aktivitäten, den verwendeten Techniken und den beteiligten Personen geklärt werden [Ru09].

Als Grundgerüst der neuen Methode dient eine Kombination des zweistufigen Prozesses von PAHL/BEITZ, und dem Vorgehensmodell nach Robertson/Robertson. Die beiden Vorgehen wurden dabei um sinnvolle Aspekte und Arbeitsschritte andere Methoden ergänzt.

Der Ablaufplan beschreibt ein siebenstufiges Vorgehen zur Erstellung einer den Forderungen des Marktsegments entsprechenden Anforderungsliste. Dabei wurden dem Prozess auf der rechten Seite die für die Bearbeitung der Prozessstufen möglichen Methoden sowie die Kernaufgaben und zugehörigen Daten im Dokument gegenübergestellt. Auf der linken Seite vom Prozess wurden möglich Inputs, die als Datenquellen dienen können, gelistet.

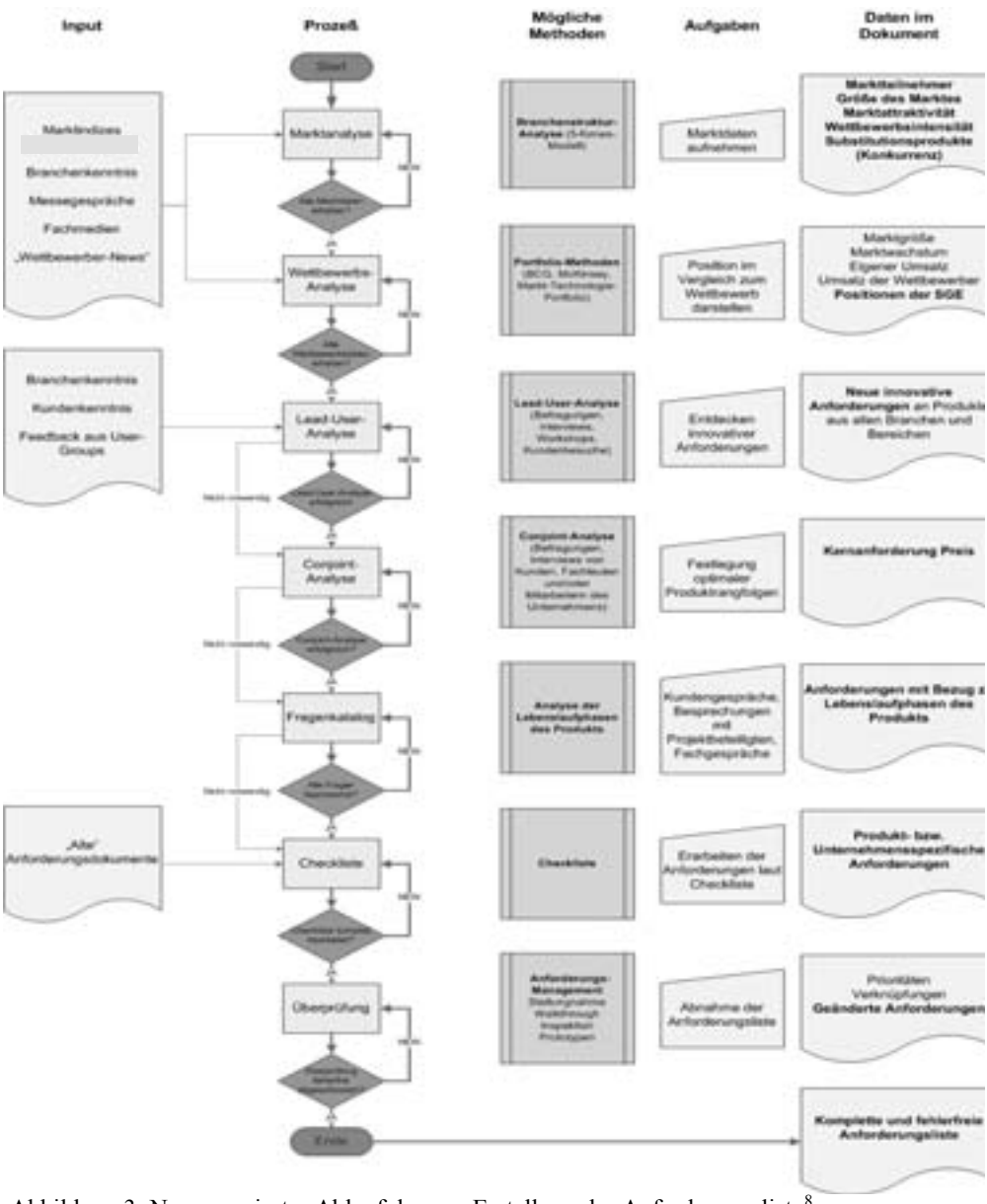


Abbildung 3: Neu generierter Ablaufplan zur Erstellung der Anforderungsliste[8]

[8] Grundsätzlich kann der entwickelte Prozess sowohl für Neuentwicklungen als auch für Variantenkonstruktionen verwendet werden. Hierbei unterliegt es dem Projektmanager benötigte Phasen zu Identifizieren. Bei der Variantenkonstruktion kann zusätzlich auf bestehende Daten und Anforderungskataloge zurückgegriffen werden, was den Aufwand für die Durchführung deutlich verringert.

Die Beantwortung dieser Fragen findet sich im Vorgehensmodell in Abbildung 3. Als hauptverantwortliche Person wird der Projektmanager angesehen. Nach dem Start des Prozesses erfolgt als erstes eine Marktanalyse, um einen Überblick über den gesamten Markt zu erhalten. Der nächste Schritt der Wettbewerbsanalyse greift auf dieselben Informationsquellen zurück. In Phase 3 erfolgt eine „Untersuchung" der Kundengruppen nach dem Lead-User Ansatz. Grundsätzlich sollten der zuständige Mitarbeiter oder andere Mitglieder des Projektteams über eine ausgeprägte Branchenkenntnis verfügen. Der nächste Schritt im Ablauf der neuen Methode ist die Conjoint-Analyse. Dabei greift die Conjoint-Analyse sowohl produktbezogenes Fachwissen der Mitarbeiter, als auch auf Kenntnisse über Wettbewerbserzeugnisse als Input zurück. Im Rahmen der Methode wird als Prozessschritt fünf ein Fragenkatalog zur Unterstützung der Erstellung von Anforderungslisten erstellt. Der sechste Schritt der neuen Methode sieht die Bearbeitung einer technischen branchenspezifischen Checkliste vor. Diese sollte die üblichen Punkte der Produktentwicklung abdecken und wiederkehrende Anforderungen an die Produkte der Unternehmen beinhalten. Zur Erstellung wird die Anforderungsschablone nach Rupp verwendet (vgl.Ru09]). Im Rahmen der Überprüfung, welche die letzte Phase der Methode darstellt, werden fehlerhafte Anforderungen entdeckt. Sie stellt die Qualitätskontrolle vor Übergabe der Anforderungsliste an die nachfolgenden Abteilungen dar.

4 Diskussion der Ergebnisse

Zur Validierung[9] des Vorgehensmodells wurde ein Prozess zur Anforderungserhebung exemplarisch in einem Projekt durchgeführt, in dem die Anforderungsanalyse zuvor bereits nach dem „alten" Verfahren durchlaufen wurde. Dies soll eine bessere Vergleichbarkeit der Ergebnisse ermöglichen. Mit der Validierung soll die Durchführbarkeit und Anwendungsfähigkeit der neuen Methode belegt werden. Hierbei wurden mittels einer Befragung der hauptverantwortlich am Prozess beteiligten Mitarbeiter nachfolgend beschriebene Ergebnisse erzielt. Die befragte Gruppe setzte sich dabei aus 10 Produktmanagern, drei Prozessplanern, vier Entwicklern/Konstrukteuren, einem Controller und einem Projektplaner zusammen.

1. Die Methode ist zur Anforderungsanalyse und –spezifikation geeignet.
2. Die Eignung für das Produktsegment ist gegeben.
3. Die neue Methode liefert gegenüber der alten Vorgehensweise qualitativ bessere Ergebnisse.
4. Das Verfahren ist aufwändiger als das bisherige Vorgehen.

[9] Unter Validierung ist der formelle und dokumentierte Nachweis der Gültigkeit einer analytischen Methode zu verstehen. Es ist zu beweisen, dass die Methode für den Einsatzzweck geeignet ist und die an sie gestellten Anforderungen erfüllt.

Die neue Methode wird als geeignetes Werkzeug zur Anforderungsanalyse gesehen und erfüllt die 10 gestellten Anforderungen. Nachteilig ist jedoch der hohe Aufwand für die Durchführung. Entscheidend ist, ob dieser wahrgenommene Mehraufwand sich in besseren Ergebnissen der Anforderungsanalyse widerspiegelt. Dieser wäre gerechtfertigt, sobald durch die intensivere Analyse der Kundenanforderungen Nachbesserungen im späteren Entwicklungsprozess vermieden würden.
Der Aufbau der Methode hat gezeigt, dass auch oder gerade Teile des Requirements Engineering, wie z.B. die Überprüfung von Anforderungen, zielführend zur Anforderungsanalyse eingesetzt werden können. Ergänzt wurden die beiden Vorgehensmodelle dabei um unterstützende Methoden und Analysemodelle, die zur Erfüllung aller Anforderungen des Produktsegments beitragen. Dabei bietet der Ablaufplan sowohl für Varianten-, als auch für Neukonstruktionen ein strukturiertes und anpassbares Vorgehen.

5 Zusammenfassung und Ausblick

Wie die Validierung und Bewertung gezeigt hat, bietet die neu entwickelte Methode zur Anforderungsanalyse und - spezifikation eine gute Ausgangsbasis zur Durchführung der Analysen für das betrachtete Produkt. Hierbei wurde ausgehend von den Modellen der PE und dem Requirements Engineering ein Ablaufplan geschaffen, der ein systematisches Vorgehen zur Ermittlung von Anforderungen bietet. So hat der Aufbau der Methode gezeigt, dass insbesondere Teile des Requirements Engineering, wie z.B. die Überprüfung von Anforderungen, zielführend zur Anforderungsanalyse in der klassischen PE eingesetzt werden können. Ergänzt wurden die beiden Vorgehensmodelle um unterstützende Methoden und Analysemodelle, die zur Erfüllung aller Anforderungen des Produktsegements beitragen. Dabei bietet der Ablaufplan sowohl für Varianten-, als auch für Neukonstruktionen von industriellen Steckverbindern ein strukturiertes und anpassbares Vorgehen. Um dieses neue Vorgehen in der Abteilung zu implementieren sollte ein Handbuch erstellt werden, welches den gesamten Ablauf der Methode anhand praxisnaher Beispiele und produktspezifischer Gegebenheiten beschreibt.

Das Vorgehensmodell der PE sowie das Modell des Requirements Engineering bieten somit gute Ansätze zur Durchführung der interdisziplinären Anforderungsanalyse und –spezifikation für das betrachtete Produktsegment. Es wird deutlich, dass für eine produktspezifische Systematik zur Anforderungsanalyse, Synergieeffekte durch ein Zusammenspiel der beiden Disziplinen erzielt werden können. Die Methoden des SE und der PE in der idealtypischen Unterscheidung sind in der Praxis wohl kaum anzufinden. Eine klare Abgrenzung der beiden Verfahren ist aufgrund einer hohen Überschneidungsrate, insbesondere auch bei den eingesetzten Tools kaum möglich. Es bleibt festzuhalten, dass für die zukünftige Methodenentwicklung der Anforderungsanalyse beide Verfahren eine starke Rolle spielen. Auch die Untersuchung der Integration weiterer Disziplinen wie der psychologischen Perspektive oder der Perspektive des Qualitätsmanagements sind zu prüfen. Ein Ziel der Weiterentwicklung und Untersuchung wären unternehmens- oder gar branchenspezifische Lösungen durch einen interdisziplinären Ansatz zu verbessern. Hierbei sollte insbesondere an der Aufwandsreduzierung für die Durchführung einer solchen Methode gearbeitet werden.

Literaturverzeichnis

[SBA02] Specht, G.; Beckmann, C.; Amelingmeyer, J.: F&E-Management – Kompetenz im Innovationsmanagement; Verlag Schäffer-Poeschel; Stuttgart; 2002

[Bu94] Bullinger, H.-J.: Einführung in das Technologiemanagement; Modelle, Methoden, Praxisbeispiele; Teubner Verlag; Stuttgart, 1994

[GPW10] Gausemeier, Jürgen; Plass, Christoph; Wenzelmann, Christoph (2009): Zukunftsorientierte Unternehmensgestaltung. Strategien, Geschäftsprozesse und IT-Systeme für die Produktion von morgen.

[vHi86] Von Hippel, E. (1986), "Lead Users: A Source of Novel Product Concepts", Management Science 32(7): 791–806

[IEEE830] IEEE Std 830-1998 - IEEE Recommended Practice for Software Requirements Specifications, IEEE Press, Piscataway, New Jersey, 1998.

[ISO9000] ISO 9000: Qualitätsmanagementsysteme - Grundlagen und Begriffe

[PBFG07] Pahl, Gerhard; Beitz, Wolfgang; Feldhusen, Jörg; Grote, Karl-Heinrich; Pahl-Beitz: Konstruktionslehre. Grundlagen erfolgreicher Produktentwicklung ; Methoden und Anwendung. 7. Aufl. Berlin: Springer (Springer-Lehrbuch).2007

[Po08] Pohl, K.: Requirements engineering. Grundlagen, Prinzipien, Techniken. dpunkt-Verl., Heidelberg, 2008.

[PR10] Pohl, Klaus; Rupp, Chris : Basiswissen Requirements Engineering. Aus- und Weiterbildung zum "Certified Professional for Requirements Engineering" ; Foundation Level nach IREB-Standard.. Heidelberg: dpunkt-Verl.2010

[RR99] Robertson, S.; Robertson, J.: Mastering the requirements process. Addison-Wesley, Harlow, 1999.

[RR06] Robertson, S.; Robertson, J.: Mastering the Requirements Process; Addison-Wesley; 2006

[Ru09] Rupp, Chris : Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis. München: Hanser.2009

[VDI2221] VDI: VDI-Richtlinie 2221 : Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte, 1993

[VDI2519] VDI: VDI-Richtlinie 2519 - Vorgehensweise bei der Erstellung von Lasten-/Pflichtenheften, 2001.

[La07] Laplante, P. (2007). What Every Engineer Should Know about Software Engineering. Boca Raton

[IEEE90] IEEE Std 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology

[Eh09] Ehrlenspiel, K.: Integrierte Produktentwicklung; Denkabläufe, Methodeneinsatz, Zusammenarbeit; 4., überarbeitete Auflage; München 2009

[Ro00] Roth, K.: Konstruieren mit Konstruktionskatalogen; Konstruktionslehre; Springer Verlag Berlin, Heidelberg,New York u.a.; 2000

[KKB07] Kotler, P.; Keller, K. L.; BLIEMEL, FRIEDHELM: Marketing Management, Strategien für wertschaffendes Handeln; Verlag Pearson Studium; München; 2007

[No06] Noe, M.: Projektbegleitendes Qualitätsmanagement: Der Weg zu besserem Projekterfolg; Publicis Verlag; Erlangen; 2006

[Po80] Porter, M. E.: Competitive Strategy: Techniques for analyzing industries and competitors; Free Press; New York; 1980

[Zu95] Zultner, R. E.: Blitz QFD: Better, Faster, and Cheaper Forms of QFD. In: American Programmer; ; S. 25-36, 1995

Effiziente Entscheidungsunterstützung im Krisenfall durch interaktive Standard Operating Procedures

T. Becker, B.-S. Lee, R. Koch

Computeranwendung und Integration in
Konstruktion und Planung – C.I.K.
Universität Paderborn
Warburger Straße 100
33098 Paderborn
E-Mail: t.becker@cik.upb.de

Abstract: Führungskräfte in Krisensituationen sind auf Grund hohen Zeitdrucks und immer komplexer werdender Situationen einem enormem Stress ausgesetzt und benötigen daher häufig eine sinnvolle Entscheidungsunterstützung. Bestehende SOPs können dieser Herausforderung aufgrund der sehr dynamischen und komplexen Einsatzabläufe nicht mehr gerecht werden. Diese Arbeit stellt ein neues Konzept *interaktiver* SOPs vor, die einen standardisierten Einsatzablauf sicherstellen, der Führungskraft dabei aber Freiraum für eigene Entscheidungen zugestehen und flexibel auf dynamische Situationen reagieren, um so eine optimale Entscheidungsunterstützung in zeitkritischen Situationen zu realisieren.

1 Motivation

Einsatzkräfte des Bevölkerungsschutzes und der Katastrophenhilfe werden vor immer größere Herausforderungen gestellt. Insbesondere Führungskräfte sind auf Grund hohen Zeitdrucks, zunehmend komplexer werdender Situationen und hohen Medieninteresses einem enormem Stress ausgesetzt und benötigen daher häufig Entscheidungsunterstützung und standardisierte Vorgehensmodelle [FEM99]. Standard Operating Procedures (SOP) als Instrument der Entscheidungsunterstützung in komplexen und sicherheitsrelevanten Situationen sind in der Medizin und in verwandten Bereichen schon seit langem bekannt. Im Rahmen des Forschungsprojekts I-LOV[1] wird untersucht, in wie weit sich Standard Operating Procedures auf die Domäne der zivilen Gefahrenabwehr anwenden lassen. Am Beispiel der „5 Phasen der Ortung und Bergung von Verschütteten“ nach einem Gebäudeeinsturz wurden in Zusammenarbeit mit dem

[1] Dieses Paper stellt ein Teilergebnis der Forschungsarbeiten der Universität Paderborn aus dem Arbeitspaket 2 im Rahmen des Forschungsprojekts I-LOV vor. Das Projekt wird vom BMBF im Rahmen der „Hightech-Strategie: Forschung für die zivile Sicherheit“ gefördert, Förderkennzeichen 13N9759 bis 13N9772. Die Autoren danken dem BMBF und allen Projektpartnern, insbesondere des Arbeitspakets 2 und dem THW, für ihre Unterstützung.

Technischen Hilfswerk 15 SOPs entwickelt, die den gesamten Einsatzablauf abbilden. Ziel ist jedoch nicht nur, die Prozesse der Ortung und Bergung zu standardisieren, sondern diese auch flexibel zu gestalten. Diese Arbeit zeigt, dass herkömmliche SOPs dieser Herausforderung nicht gerecht werden können (Kapitel 2). Stattdessen wird in Kapitel 3 ein neues Konzept *interaktiver* SOPs vorgestellt, die einen Kompromiss aus standardisierten Handlungsanweisungen und flexiblen, hochdynamischen Prozessen bilden. In Kapitel 4 wird am Beispiel von I-LOV die praktische Anwendung interaktiver SOPs und deren Einbettung in ein Gesamtsystem zur Erhöhung der zivilen Sicherheit beschrieben.

2 Standard Operating Procedures – Begriff und Stand der Technik

Der Begriff „Standard Operating Procedures“ ist in der bestehenden Literatur nicht eindeutig definiert oder er wird von verschiedenen Autoren unterschiedlich angewendet. Gemeinhin versteht man unter Standard Operating Procedures niedergeschriebene organisatorische Anweisungen, die eine standardisierte und erprobte Vorgehensweise bei häufig wiederkehrenden Prozessen explizit vorschreiben oder empfehlen. Häufig werden SOPs für administrative Prozesse (Personalwesen, Wartung und Instandhaltung, Arbeitssicherheit und Umweltschutz etc.) und Prozesse der Gefahrenabwehr und -beseitigung (Feuerbekämpfung, medizinische Hilfe, Gefahrstoffumgang etc.) geschrieben. Diese SOPs helfen Entscheidungsträgern in Krisensituationen oder komplexen betriebswirtschaftlich-technischen Situationen, immer schwierigere Aufgaben mit gesteigerten Anforderungen bei geringeren Budgets, personellen und materiellen Ressourcen zu bewältigen. Darüber hinaus tragen SOPs zu einer Standardisierung der Kernprozesse einer Organisation bei und sichern so bestehende Qualitätsniveaus. Standard Operating Procedures können auch zur effektiven Ausbildung und zum Training von Einsatzkräften verwendet werden. Zu den weiteren Vorteilen zählen u. a. mehr Sicherheit, höhere Effizienz und dadurch auch ökonomische Vorteile. [FEM99], [Sig08]

Standard Operating Procedures stammen ursprünglich aus dem medizinischen Bereich (dort lassen sich sehr viele Beispiele und Veröffentlichungen finden, die unterschiedliche Aufgaben und Behandlungspfade betrachten, z. B. [Sig08], [MSK03], [BH08]). Aber auch in verwandten Domänen wie der Biologie und Chemie oder Laborarbeit im Allgemeinen [Ree01], [Web01], [Web02] sowie in der Luft- und Raumfahrt [Web03] zählen SOPs zu den gebräuchlichen Werkzeugen. Es gibt auch schon erste Ansätze im Bereich der zivilen Gefahrenabwehr. Diese kommen hauptsächlich aus den USA. Zum Beispiel ist das Oklahoma City Fire Department sehr engagiert, große Einsätze unter dem Motto „Lessons Learned“ aufzubereiten und diese in Form von SOPs für zukünftige Anwendungen zu dokumentieren [Lee05]. Es existieren viele Handbücher und Leitfäden zum Schreiben effektiver Standard Operating Procedures, die eine ausführliche Hilfestellung bei der Erstellung, Einführung und Evaluation von SOPs geben [RB06], [EPA07]. Alle Quellen haben jedoch eins gemeinsam: Sie betrachten SOPs als Klartext-Prozessbeschreibungen. Die meisten SOPs haben einen Umfang von 5 bis 20 DIN-A4-Seiten, einige gehen sogar darüber hinaus. Das macht sie zu wertvollen „Standard Operating Guidelines“, wie sie gelegentlich auch

genannt werden, das heißt zu Richtlinien, die zur Einsatzvorbereitung, zur Schulung oder für Übungen genutzt werden können. Aber für eine effiziente Entscheidungsunterstützung in einem realen Krisenfall sind diese „Guidelines“ zu umfangreich. Während eines echten Einsatzes ist die verfügbare Zeit in der Regel sehr knapp. Der Einsatzleiter bzw. ein Abschnittsleiter hat in einer angespannten Lage nicht ausreichend Zeit, um mehrere Seiten eines Handbuchs zu lesen. In Deutschland werden diese Dokumente als „Standard-Einsatz-Regeln (SER)“ bezeichnet [GCV03]. Auch sie erstrecken sich über durchschnittlich 5 bis 10 Seiten Klartext.

Wesentlich effizienter sind dagegen SOPs, die als Flussdiagramm oder Checkliste dargestellt werden. Beide Darstellungsformen ermöglichen einen schnellen Überblick über alle notwendigen Arbeitsschritte, die noch durchzuführen sind. Die Checkliste als eine Reihe von Arbeitsanweisungen, die einfach abgehakt werden, ist sehr übersichtlich und einfach zu handhaben. Sie ist ideal für starre Prozesse, die immer in der gleichen Reihenfolge ablaufen: Arbeitsschritt 1, gefolgt von Arbeitsschritt 2, gefolgt von … Arbeitsschritt n. Sobald allerdings Flexibilität verlangt wird, stößt die Checkliste schnell an ihre Grenzen. Ist die Reihenfolge abhängig von äußeren Parametern, muss mit Verweisen gearbeitet werden: „Wenn Bedingung A erfüllt, weiter mit Arbeitsschritt 3, sonst weiter bei Arbeitsschritt 9“. Bei komplexeren Prozessen mit mehreren Parametern ist die Checkliste nicht mehr hilfreich. Etwas mehr Flexibilität bietet das Flussdiagramm, da dieses grafisch arbeitet und so multiple Verweise ermöglicht. Allerdings ist auch hier ersichtlich, dass sich bei umfangreicheren Prozessen das Flussdiagramm schnell auf mehrere DIN-A4-Seiten erstreckt, die Verweise über mehrere Seiten gehen und die Führungskraft durch häufiges Umblättern und Suchen nicht mehr wirklich effizient unterstützt wird. Diese Form der SOPs kann also der Herausforderung dynamischer und komplexer Prozesse nicht gerecht werden.

3 Interaktive SOPs

Einen Kompromiss aus hoher Standardisierung einerseits und einem Maximum an Flexibilität bei einfacher Handhabung andererseits sollen *interaktive* Standard Operating Procedures bieten. Durch eine geeignete IT-Unterstützung sollen die Vorteile der Checkliste, des Flussdiagramms und der herkömmlichen, textbasierten SOP vereint werden. Interaktive Standard Operating Procedures sollen einen standardisierten Einsatzablauf sicherstellen, der Führungskraft dabei aber Freiräume für eigene Entscheidungen und Erfahrungen zugestehen und flexibel aber dennoch schnell auf dynamische Situationen reagieren, um so eine optimale Entscheidungsunterstützung in zeitkritischen Lagen zu realisieren.

3.1 Lösungsansatz: Funktion und Darstellung von interaktiven SOPs

Um diesen Anspruch zu erfüllen, werden die interaktiven SOPs in Form von Checklisten dargestellt, die sich aus Arbeitsanweisungen, Abfragen und Hinweisen (bei akuten Gefahren: Warnungen) zusammensetzen. Jede SOP besteht aus einer oder aus einer Folge von Checklisten, die eine Menge von Arbeitsschritten sinnvoll gruppieren und

nacheinander abzuarbeiten sind. Auf diese Weise bekommt der Benutzer immer nur die zurzeit relevanten Arbeitsanweisungen angezeigt und entscheidet, ob diese Arbeitsschritte durchgeführt werden sollen oder nicht[2]. Durch die Beantwortung der Abfragen tritt der Benutzer mit dem IT-System in eine Interaktion. Abfragen können simple Ja/Nein-Entscheidungen beinhalten, konkrete Werteingaben (Strings, Integers, Floats etc.) erfordern oder Auswahlfelder anzeigen. Je nachdem, welche Eingaben der Benutzer in das IT-System einträgt, wird die darauffolgende Checkliste dynamisch zusammengestellt. So wird jede SOP an die aktuelle Situation angepasst. Es werden dieser entsprechend nur sinnvolle Arbeitsanweisungen angezeigt. Diese Methode gewährleistet einen standardisierten Einsatzablauf (entsprechend der hinterlegten SOP-Logik, siehe 3.3) und standardisierte Arbeitsanweisungen (SOP-Inhalte), aber durch die Interaktivität auch ein hohes Maß an Flexibilität.

Als Ergänzung zu den Checklisten, die lediglich kurze und präzise Arbeitsanweisungen bereitstellen, können zu jedem Arbeitsschritt Zusatzinformationen hinterlegt werden, die nur bei Bedarf angezeigt werden, sei es in Form von integrierten Hinweistexten oder in Form von angehängten Dokumenten (Handbücher, Grafiken, Tabellen etc.). Ein weiterer wichtiger Bestandteil von interaktiven SOPs im Katastrophenfall sind Ereignisse. Ereignisse sind plötzlich auftretende Situationsveränderungen, die eine schnelle Reaktion erfordern. Beispiel: Wird in Gebäudetrümmern ein Hohlraum entdeckt, muss der aktuelle Prozess unterbrochen werden und der Hohlraum auf mögliche Verschüttete hin untersucht werden. Sobald der Benutzer den Ereignisbutton „Hohlraum gefunden“ anklickt, wird automatisch, ohne Verzögerung, die SOP „Ortung“ gestartet.

3.2 Definition einer Algebra für interaktive SOPs

Der Aufbau von interaktiven SOPs orientiert sich zunächst an der gewohnten Beschreibung eines Geschäftsprozesses, bestehend aus einer textuellen und einer grafischen Darstellung der SOP. Der Text beschreibt die Einbettung der SOP in die gesamte Prozesslandschaft: Ziel und Nutzen der SOP, Ein- und Ausgaben, Schnittstellen zu anderen SOPs usw. Ein Flussdiagramm stellt den Ablauf der SOP anschaulich dar: Arbeitsanweisungen, Abfragen, Hinweise, deren Gruppierung zu Checklisten etc. (SOP-Inhalte) und die Verweise und Bedingungen (SOP-Logik).

Um standardisierte SOPs zu erstellen und diese später in einem IT-System zu implementieren, z. B. in einer Datenbank zu speichern, wurde eine simple Algebra definiert. Die Abbildung 1 beschreibt einen Ausschnitt aus dem schematischen Aufbau einer interaktiven SOP: Jede SOP besteht aus einer oder aus mehreren Checklisten, die jeweils durch zwei Separatoren begrenzt werden (Start und Ende der Checkliste). Innerhalb einer Checkliste können Arbeitsanweisungen und Fragen (Checklisteneinträge) sowie Hinweise bzw. Warnungen definiert werden, die später über die GUI ausgegeben werden. Bei jedem Checklisteneintrag kann der Benutzer dazu aufgefordert werden, eine oder mehrere Eingaben zu machen. Diese Inputs können unter

[2] Da ein IT-System lediglich die Entscheidungsfindung unterstützen aber niemals die Verantwortung für die vorgeschlagenen Arbeitsschritte übernehmen kann, bleibt die Verantwortung beim Einsatz- bzw. Abschnittsleiter und damit auch die Entscheidung, ob ein Arbeitsschritt durchgeführt wird oder unnötig ist.

einem definierten Variablennamen entweder für eine spätere Verwendung abgespeichert oder auch direkt in der folgenden Checkliste ausgewertet werden. In diesem Fall bildet die Nutzereingabe die Grundlage für eine Systemabfrage. Dies kann eine simple Ja/Nein-Entscheidung sein oder auch eine verschachtelte Abfrage, eine Fallunterscheidung usw. (in der Grafik nicht dargestellt). Je nach Bedingungen und Ausprägungen der Variablen folgt ein anderer Checklisteneintrag.

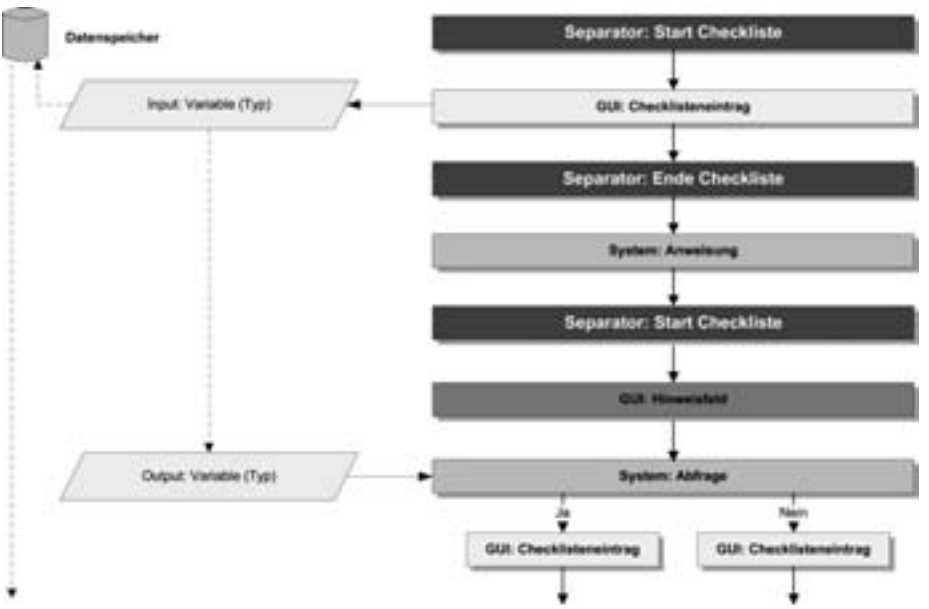


Abbildung 1: Ausschnitt aus dem Aufbauschema einer interaktiven SOP

Für den Fall, dass eine simple Logik nicht ausreicht, können in einer SOP Systemanweisungen platziert werden, die einen zusätzlichen Programmcode aufrufen. Hier können beispielsweise komplexe Berechnungen durchgeführt oder eine automatische Einsatzdokumentation geführt werden. Über diese Schnittstelle können die Funktionalitäten des SOP-Moduls zusätzlich erweitert werden. Systemanweisungen können aber auch Startbefehle für neue SOPs sein.

Jede interaktive SOP eines Einsatzszenarios wird zunächst textuell beschrieben und mit den Elementen dieser Algebra definiert. Im Grunde unterscheidet sich diese Beschreibung inhaltlich nicht von herkömmlichen SOPs. Der eigentliche Mehrwert, d. h. eine leichte und effiziente Handhabung, wird erst durch die folgende Implementierung erzielt.

3.3 Implementierung der interaktiven SOPs

Für die Speicherung in einer Datenbank werden die SOPs in Logik und Inhalt unterteilt. Die Inhalte, d. h. die Beschreibung der einzelnen Checklisteneinträge, werden einmalig definiert und können bei mehrfacher Verwendung in verschiedenen SOPs[3] an unterschiedlichen Stellen eingefügt werden. In der logischen Struktur werden dann Verweise auf diese Checklisteninhalte gesetzt, durch eine serielle Verknüpfung in eine sinnvolle Reihenfolge[4] gebracht und zu individuellen Checklisten gruppiert. Durch die Wiederverwendung von Inhalten werden Redundanzen innerhalb der Datenbank vermieden und Formulierungen standardisiert. Alle aus dieser Untergliederung resultierenden Objekte (SOPs, Checklisten, Arbeitsanweisungen, Abfragen, Hinweise, Zusatzinformationen, verschiedene Eingabetypen, Ereignisse etc.) wurden in einem ER-Modell bzw. Klassendiagramm abgebildet. Die logische Struktur wird durch Relationen zwischen den Objekten realisiert. Durch die Verwendung von Referenzen auf Variablen, deren Werte der Benutzer in vorherigen Checklisten oder SOPs eingegeben hat, werden diese Relationen mit zusätzlichen Bedingungen angereichert.

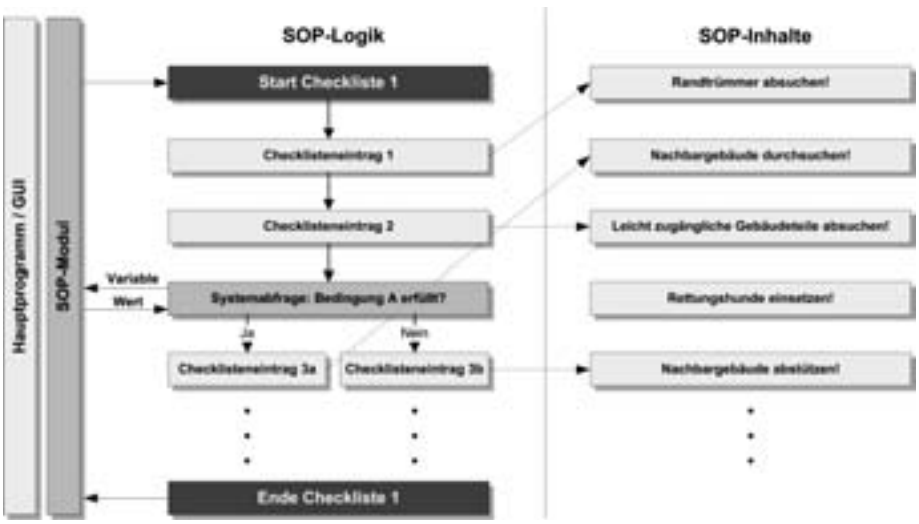


Abbildung 2: Logik und Inhalt einer interaktiven SOP

Während eines Einsatzes liest ein Algorithmus die logische Struktur aus der Datenbank, beginnend mit dem ersten Checklistenobjekt einer SOP. Über einen Verweis auf den

[3] Beispielsweise wird die Aufforderung, Sicherungsvorkehrungen zu treffen, über den gesamten Einsatz hinweg an vielen Stellen erscheinen und Teil mehrerer SOPs sein.
[4] Auch wenn die Arbeitsschritte innerhalb einer Checkliste zwangsläufig in einer linearen Reihenfolge dargestellt werden müssen, können diese in der Praxis durchaus parallel bearbeitet werden.

entsprechenden Inhalt wird die konkrete Arbeitsanweisung eingelesen. Der Algorithmus folgt anschließend der Verknüpfung zum jeweils folgenden Objekt, arbeitet sich so bis zum Ende einer Checkliste voran und fügt die einzelnen Inhalte zu einer Checkliste zusammen, die über die GUI des Hauptprogramms ausgegeben wird (siehe Abbildung 2). Trifft der Algorithmus auf eine bedingte Verknüpfung, gelangt er über eine Relation zum folgenden Objekt. Je nach Ausprägung der abgefragten Variablen zeigt die Verknüpfung auf einen anderen Checklisteneintrag. Beispielsweise sollte ein Nachbargebäude nur durchsucht werden, wenn es nicht einsturzgefährdet ist (Bedingung A erfüllt), andernfalls ist dieses erst abzustützen. So ergeben sich multiple Pfade. Je nach Benutzereingaben werden die Checklisten unterschiedlich zusammengestellt.

4 Interaktive SOPs im Forschungsprojekt „I-LOV“

Im vorigen Kapitel wurde das reine SOP-Modul beschrieben, welches eine Art Backend-Service mit eigener SOP-Datenbank darstellt. Nun soll auch die praktische Anwendung kurz vorgestellt werden: Im Forschungsprojekt I-LOV wird das Entscheidungsunterstützungssystem „SOPHIE“ entwickelt. Es soll bei einem Gebäudeeinsturz insbesondere das Technische Hilfswerk (THW) durch Informationsverdichtung, interaktive Standard Operating Procedures, eine digitale Lagekarte und durch die Integration neuer Technologien bei der Ortung von Verschütteten unterstützen. Die Entwicklung des zentralen IT-Systems ist ein gemeinsames Teilprojekt mehrerer Projektpartner (Teilprojekt 2). Java und OSGi ermöglichen eine plattformunabhängige und modulare Entwicklung mit definierten Schnittstellen. Das oben beschriebene SOP-Modul bildet einen Teil von SOPHIE und stellt einen Service für die dynamische Generierung interaktiver SOPs und die Verarbeitung der Nutzereingaben bereit. Die grafische Ausgabe der Checklisten erfolgt über die GUI des Hauptprogramms.

Der Einsturz eines Gebäudes ist ein wiederkehrendes Szenario, in dem das THW häufig die Ortung und Bergung von Verschütteten übernimmt. Die häufigste Ursache sind Gasexplosionen. Aber auch unter eingestürzten Dächern durch schwere Schneemassen[5] können Menschen verschüttet werden. Der Einsatzablauf orientiert sich an den sogenannten „5 Phasen der Bergung“ [vfd05]. Auf deren Grundlage wurde von der Universität Paderborn in Zusammenarbeit mit dem THW unter Einbeziehung von Fachliteratur, Ausbildungsunterlagen, Übungsteilnahmen und Expertengesprächen ein Prozessmodell entwickelt, welches den gesamten Einsatzablauf, von der Ersterkundung bis hin zur Räumung der Schadenstelle, beschreibt. Aus diesem Prozessmodell heraus wurden 15 individuelle SOPs identifiziert und erstellt, die die Führungskräfte mit allen wichtigen Arbeitsanweisungen für den gesamten Einsatz mit Schwerpunkt Ortung und Bergung versorgen. Die SOPs wurden derart ausgewählt, dass in sich geschlossene Aufgabenkomplexe entstehen (z. B. Sicherung, Ortung, Bergung). Das SOP-Modul ist zurzeit vollständig implementiert und in das Entscheidungsunterstützungssystem integriert, so dass die interaktiven SOPs im Falle eines Gebäudeeinsturzes bzw. in einer vergleichbaren Übung zur Anwendung gebracht werden können.

[5] Zur Zeit der Veröffentlichung (Winter 2010/2011) ein sehr aktuelles Thema.

5 Ausblick

Durch die Einhaltung der „5 Phasen der Bergung“ und die Gestaltung der SOPs ist ein standardisierter Einsatzablauf sichergestellt. Eine effiziente Entscheidungsunterstützung ist dadurch gegeben, dass ein der Situation angemessenes und vollständiges Vorgehen empfohlen wird (Vermeidung von stressbedingten Versäumnissen, Rechtssicherheit für den Entscheidungsträger, Hinweis auf Gefahren, Integration von neuer Technologien und Best Practices, etc.), der Aufwand für den Entscheidungsträger allerdings minimal gehalten wird. Die Auswertung von Benutzereingaben und die Reaktion auf Ereignisse sollen auch einen flexiblen Einsatzverlauf ermöglichen. Ob diese Ziele durch die Verwendung von interaktiven SOPs tatsächlich erreicht worden sind, soll eine abschließende Evaluation innerhalb des Forschungsprojekts zeigen. Ebenso wird die Handhabung und Akzeptanz der SOPs und des Gesamtsystems in zeitkritischen Situationen zu untersuchen sein.

Für die Weiterentwicklung des SOP-Moduls ist die Erforschung einer kollaborativen SOP-Entwicklung (in Anlehnung an Wickler & Potter [WP10]), z. B. in Form eines SOP-Editors, geplant.

Literaturverzeichnis

[BH08] Bleyl, J. U.; Heller, A. R.: Standard Operating Procedures und OP-Management zur Steigerung der Patientensicherheit und der Effizienz von Prozessabläufen. Springer Verlag, 2008

[EPA07] United States Environmental Protection Agency (EPA): Guidance for Preparing Standard Operating Procedures (SOPs). Washington, 2007

[FEM99] Federal Emergency Management Agency (FEMA): Developing Effective Standard Operating Procedures For Fire and EMS Departments. IOCAD Emergency Services Group, 1999

[GCV03] Graeger, A.; Cimolino, U.; de Vries, H.; Haisch, M.; Südmersen, J.: Einsatz- und Abschnittsleitung – Das Einsatz-Führungs-System. ecomed Sicherheit, Landsberg, 2003

[Lee05] Lee, K. J.: Untapped Resources: Developing a Standard Operating Procedure for Support Staff Response to Significant or Large-Scale Incidents in Oklahoma City. Oklahoma City, 2005

[MSK03] Martin, J.; Schleppers, A.; Kastrup, M.; Kobylinski, C. et. al.: Development of Standard Operating Procedures in anaesthesiology and critical care medicine. In: Anästhesiologie & Intensivmedizin. DIOmed-Verlags GmbH, 2003

[Ree01] van Reeuwijk, L. P.: Guidelines for Quality Management in Soil and Plant Laboratories. Daya Publishing House, 2001

[RB06] Raetzell, M.; Bauer, M.: Standard operating procedures und klinische Behandlungspfade. In: OP-Management: praktisch und effizient. Springer Verlag, Heidelberg, 2006

[Sig08] Sigel, S. T.: Umfrage zur Entwicklung der Analgesie und Sedierung des Intensivpatienten im Jahr 2006 im Vergleich zum Jahr 2002. Dissertation, Universität Ulm, 2008

[VFD05] vfdb-Richtlinie 03/01 für Gebäudeeinstürze. Vds, Köln, 2005

[Web01] NFSMI/USDA: HACCP-Based Standard Operating Procedures.
http://sop.nfsmi.org/HACCPBasedSOPs.php, Stand 29.12.2010

[Web02] Tennessee Valley Authority: Standard Operating Procedures for environmental sampling.
http://www.tva.gov/kingston/sap/, Stand 29.12.2010

[Web03] NASA: Sample SOPs. Stand 29.12.2010
http://search.nasa.gov/search/search.jsp?nasaInclude=standard+operating+procedure

[WP10] Wickler, G.; Potter, S.: Standard Operating Procedures: Collaborative Development and Distributed Use. Proceedings d. ISCRAM, Seattle, 2010

A Service-oriented Architecture for Emergency Management Systems

Stephan Braune*, Achim D. Brucker†, Gerald Kleser†, Keqin Li†, Theo Dirk Meijler†, Heiko Paulheim†, and Florian Probst†

*B2M Software AG, Germany
StephanBraune@gmx.de

†SAP Research, SAP AG, Germany
{achim.brucker, gerald.kleser, keqin.li, theo.dirk.meijler, heiko.paulheim, f.probst}@sap.com

Abstract: The complexity and openness of today's modern societies result in the threat of serious cascading effects when natural disasters or terrorist attacks strike. Thus, there is a high demand for state of the art IT support in the field of emergency management systems.

In this paper, we identify the core requirements of future emergency management systems and present a new generation of modular, service-oriented and semantic-web-based architecture for emergency management systems. Our emergency management system offers innovative functionality in the context of distributed information sources, collaborative work environments, and consistent situation pictures.

1 Introduction

The complexity and openness of today's modern societies result in the threat of serious cascading effects when natural disasters or terrorist attacks strike. Modern information technologies have changed everyday life fundamentally, yet in the field of emergency management, such technologies have not yet been brought to their full potential. This is especially the case for handling large incidents, e. g., natural or man-made disasters that exceed the capabilities of the individual emergency response organization. For supporting such situations, IT systems must, on the one hand, support the various relief organizations, police, fire brigades, as well as other emergency organizations and private organizations. On the other hand, such systems have to integrate, in a reliable, secure, and user friendly manner, a large number of both internal and external information sources and services that operate on these information sources.

In this paper, we present a detailed list of requirements that novel emergency management systems (EMSs) should fulfill (Section 2). In Section 3, we list the architectural decisions we take to fulfill the previously identified requirements and introduce the resulting service-oriented architecture. This architecture was developed during the research project *SoKNOS* (Service-oriented Architectures supporting public security networks) and realized as the *SoKNOS EMS*. The requirements for the SoKNOS EMS have been developed together with potential end users such as fire brigades and the police, who also partici-

pated in the evaluation of the system. Finally, we give a brief overview of related EMSs (Section 4) and close with some concluding remarks (Section 5).

2 Requirements for Emergency Management Systems

Large-scale emergencies and disasters require the operation of a command structure consisting of multiple emergency management teams, where each team manages the operations of one or more of the major organizations involved, such as fire departments, police, and rescue and technical relief organizations. An EMS will therefore need to support an emergency management team in:

- *Accessing Information*, i. e., finding, exchanging, and evaluating information about an emergency faster;
- *Planning*, i. e., scheduling operations in a clear-sighted and interactive manner;
- *Collaboration*, i. e., simplifying the cooperation between organizations and experts in the public security sector;
- *Decision making*, i. e., reducing the reaction time and increasing the reliability of the actions taken;
- *Situational insight*, i. e., providing a prompt and comprehensive overview of the disaster and the rescue operation status using user-friendly information technology.

Based on user studies conducted in the SoKNOS project together with end users, we have identified a set of essential requirements to achieve a new generation of user-centric, Web service-based EMSs:

R1: *Integrating new information sources during run-time.* A central requirement of EMSs is the ability to integrate new and heterogeneous information services from various providers, such as traffic and weather information, flood warnings, or the availability of resources of other emergency response organizations. This in turn requires that the user can easily find new information services and assess the syntactic and semantic interoperability with the information services that are already integrated.

R2: *Extending the system functionality during run-time.* The nature of large disasters is that full preparation against them is hardly possible. Emergency response plans will usually be insufficient. Hence, the possibility of quick adaptation to the new situation is important. With respect to the EMS, the possibility to extend the offered functionality of a system with new functional modules must be given.

R3: *Enabling collaborative work within and across organizations.* A well known deficit in modern emergency management is the insufficient possibility for organizations to jointly plan appropriate actions and measures without sending liaison officers to the corresponding organizations. Sound IT support for jointly developing a common situational picture and, based on this picture, developing strategies and plans for fighting the disaster is missing. Hence, an important requirement is that multiple team members are enabled to share the same information and work collaboratively on this information.

R4: *Ensuring information consistency.* Information consistency is paramount for establishing a common situation picture. Incidents develop quite fast. Consequently, in current systems, the operational pictures of collaborating organizations differ considerably. Furthermore, information might get lost and inconsistencies arise which might finally lead to wrong decisions. Therefore, functionality is needed which enables to track and eliminate information inconsistency within and across organizational borders.

R5: *Tracking objects across views.* Solutions to the previous requirement introduce the requirement of visualizing the same objects in parallel in spatial, temporal and organizational views. Physical objects like roads, buildings or rescue teams are involved in processes (temporal entities). This involvement can be passive, like a dam involved in a flood, or active, like a rescue team involved in rescuing or protecting. Tracking an individual entity in parallel across these different spatial, temporal and organizational views enables the user to get a holistic and highly realistic picture of the situation at hand.

R6: *Overcoming semantic heterogeneities.* Even when operating in the same region or country, organization often use slightly different vocabulary. The requirements regarding information exchange and collaborative work introduce the problem of semantic heterogeneities within jointly used information sources. Hence, a central requirement for a new generation of EMSs is that they provide methods to overcome the problems that arise when the same word is used with different meanings, or one concept is addressed with different words.

R7: *Managing system reliability.* A failure of system parts should have minimal consequences, i. e., a failure of the EMS for one emergency management team should not spill over to other EMSs. Within a one EMS, temporary network failure between workstations should not result in system break down. Further, the failure of a (local) system component should not lead to a failure of other system components or system functions.

R8: *Managing security dynamically.* Emergency management frequently deals with information that needs to be protected against unauthorized access. Thus, a central requirement is that high safety and security standards are fulfilled. However, in severe emergencies, safety and security standards that are useful for enterprise systems can drastically hinder efficient counter measures. Hence, the access control should support exceptional situations, i. e., the controlled overruling of access control decisions should be supported [BP09]. Furthermore, the communication channels between different EMSs need to ensure the integrity, confidentiality, and non-repudiation of the transferred messages.

R9: *User friendliness via personalization of system functionality and adjustment of level of detail.* Large emergencies are exceptional events, thus, familiarity with a specialized user interface which needs thorough training cannot be assumed. Furthermore, stress is a crucial factor in the management of large incidents. The EMS should therefore minimize the user's cognitive load. The EMS should effectively avoid the "threat rigidity syndrome" where additional stress is caused because of a loss of control over the situation or reduced understanding of reality [TCdWY04]. Hence, the EMS should consider the needs of the various users and the roles they have when engaged in handling the incident.

3 Future Emergency Management Systems

In this section, we briefly explain the most important design decisions that characterize the proposed EMS architecture and depict their relation to the requirements identified above, and we discuss the SoKNOS architecture and prototype.

3.1 Design Decisions

The design decision we are presenting can be classified into design decisions regarding a *single EMS* and decisions regarding the *collaboration between several EMSs*.

3.1.1 Architectural Decisions Regarding a Single EMS

D1 (R1, R2): *Vertical decoupling of components across front-end-, back-end-, and information source layers*. The encapsulation of the visualization and user interface (front-end layer), the data processing (back-end layer), and the access to various information and data sources (data source layer) allows several visualizations of the same back-end services.

D2 (R1, R2, R7): *Horizontal decoupling of components on front-end-, back-end- and information source layers.* The horizontal decoupling, i. e., modularization within each layer, allows for each organization to customize the EMS they use.

D3 (R2, R5): *Loosely coupled component collaboration through event-driven publish and subscribe.* Using an event-driven publish and subscribe service between the different components (both within one layer and across layers) enables the flexible extensibility of the EMS with new functionalities (e. g., new visualizations, new processing services, or additional information services). On the front-end layer, a reasoner implements semantic event processing between UI components and back-ends and serves as an indirection for ensuring loose coupling.

D4 (R1, R2): *Configuring and maintaining the functional components during run-time.* Supporting a flexible mechanism for configuring and maintaining the various components of an EMS at runtime allows for customizing the EMS for different usage scenarios (e. g., different tasks or different user preferences).

D5 (R1, R5): *Storing data relations between independent services separately.* Using a dedicated component for storing the data relations between the different services allows for integrating new data sources implemented as services without the need of modifying the (internal) representation of the service being integrated.

D6 (R4, R6): *Using semantic information as glue between components.* The continuous use of semantic annotations, both across layers and within each layer, allows for seamless interplay of decoupled components even though they have to cope with various heterogeneous types of data and functionalities.

D7 (R5, R9): *Establishing a single point of interaction.* Integrating all user interface components into a homogeneous user interface framework providing a single interaction point

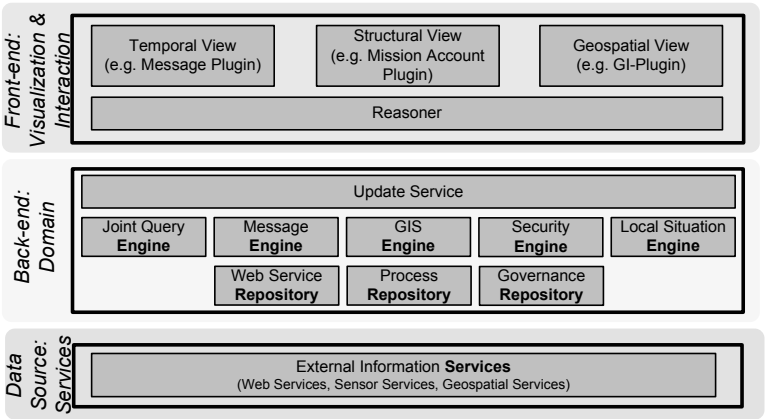


Figure 1: High-level view of the SoKNOS architecture, consisting of three decoupled layers: the data source layer, the back-end layer, and the front-end layer.

avoids the risk of distracting users by heterogeneous user interfaces and usage paradigms.

3.1.2 Architectural Decisions Regarding Collaboration between Several EMSs

D8 (R7): *Each emergency management team operates an individual independent EMS.* To ensure a high reliability even when faced with an unstable infrastructure, we assume an EMS is based on a distributed architecture (in contrast to a centralized one, e. g., using a central data storage for several EMSs). Thus, every EMS can operate (providing core functionality) in isolation, e. g., without network connectivity.

D9 (R3, R4, R9): *Facilitating cross-system information exchange.* To ensure the timely and efficient exchange of information and services between different EMSs, we apply the same principles across systems as we use for the collaboration between the components of one single EMS.

D10 (R8): *Using a decentralized identity and access control management.* On the one hand, EMSs handle sensitive data that needs to be controlled strictly by the organization operating a specific EMS. On the other hand, the efficient handling of an emergency requires the open and efficient exchange between different emergency management teams that might be operated by different organizations. A decentralized management of the security infrastructure (including, among others, the identity management and the access control configuration) enables the reliable, flexible, and secure handling of sensitive data.

3.2 Overview of the SoKNOS EMS Architecture

Following the design decisions outlined in the previous section results in a flexible, service-oriented architecture for EMSs, which we call the *SoKNOS EMS Architecture*. Figure 1 illustrates the SoKNOS EMS architecture, consisting of three decoupled layers, from bot-

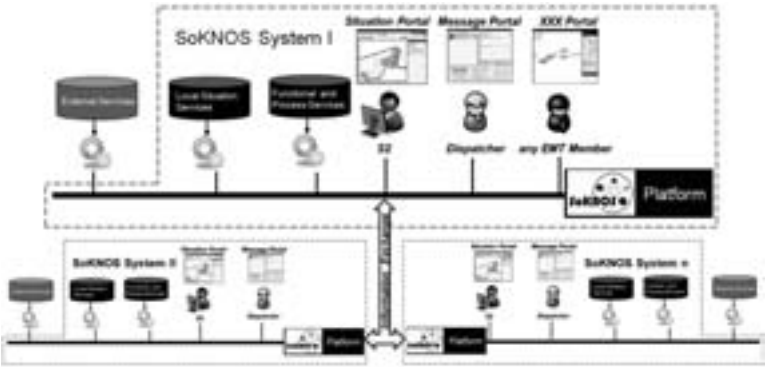


Figure 2: Each SoKNOS EMS is a distributed system; the components are connected via the SoKNOS platform. SoKNOS EMSs can be connected via the same platform functionality that is used to connect components of a single SoKNOS system.

tom to top: the data source layer, the back-end layer, and the front-end layer. In each of the layers, Figure 1 provides also a brief summary of the most important components for handling the different functional and non-functional requirements an EMS has to fulfill.

To our knowledge, the SoKNOS EMS Architecture is the first proposal for a distributed, highly dynamic EMS which makes heavy use of semantic web technologies. Figure 2 shows this important aspect: SoKNOS EMSs are themselves distributed systems, and multiple SoKNOS EMSs being used by different emergency management teams can collaborate on basis of the information exchange between these systems. This information exchange is based on the same platform approach that is used within each SoKNOS EMS.

3.3 Overview of the SoKNOS EMS Prototype

During the SoKNOS project, we have implemented a prototype that provides an EMS implementing the SoKNOS EMS Architecture. As an underlying service framework, also providing the functionalities required for the collaboration of several SoKNOS EMSs in a distributed manner, this prototype is based on Apache Tuscany [PKA+10]. Our prototype integrates various internal and external services that allow, for example, the efficient information gathering, simulation and planning, and thus supports the work of a local emergency management team. Moreover, our prototype supports the efficient and secure collaboration between emergency management teams either using the SoKNOS EMS or working with other systems.

On the one hand, we are using this prototype for evaluating the SoKNOS approach in collaboration with the German fire brigades and police forces. On the other hand, this prototype is presented in a “Living Lab” at SAP Research, showing the benefits of modern IT in the public security in safety sector to a widespread audience.

4 Related work

Several commercial and open source IT systems exist in the area of emergency management. Most of them, e. g., DISMA [TUV], Sahana [CDSDS+07], and deNIS [DEN11], are mainly deployed in control centers. They are based on a centralized system architecture and thus require a reliable and fast infrastructure. While there are systems that are operated locally within one emergency management team, such as the IGNIS system [IGN06] used in the Berlin Fire Department, they do not support the efficient collaboration across several emergency management teams.

A lot of research is currently being performed for improving emergency management systems, e. g., in projects such as ALARM, LAGE, or MANET (see [Bun10] for a survey). They concentrate on various aspects, such as performing simulations, mobile, ad hoc, and sensor-based networks, and data aggregation and visualization [Und10].

To our knowledge, the SoKNOS EMS architecture is the first attempt to provide decentralized, semantic-web-based EMS that is designed with simplifying the communication and collaboration between several organizations in mind, especially that the SoKNOS EMS architecture focuses more on the flexible integration of different functionalities and data and processing resources.

5 Conclusion

Emergency management systems raise complex requirements that, at least in this combination, are seldom required by today's enterprise systems. However, the conclusions that can be drawn from designing emergency management systems are also valuable for future enterprise systems. For these future enterprise systems, we predict a similar combination of contrary requirements. This prediction is based on our observation that the modern enterprises need to participate in highly dynamic supply chains and, for example, exchange information even with their direct competition. Based on our experience in the emergency management domain, we conclude that service-oriented systems profit from the following fundamental concepts:

- *Decentralized systems:* Decentralized systems provide, compared with centralized systems, a higher reliability combined with a higher flexibility on choosing integrating local services. Moreover, decentralization simplifies the implementation of the strict data separation requirements [BH10].
- *Decoupling through publish-subscribe:* The loose coupling achieved by the publish-subscribe model allows for building flexible systems that, e. g., can quickly integrate new information sources or services.
- *Decoupling through ontologies:* Ontologies, as a machine-readable information representation, enable the effective collaboration across different organizations, as well as across independently developed components [PP10].

For the future, we see two main lines of work that build on top of each other: We want to further focus on the collaborative aspects of the SoKNOS EMS systems: how to share processes, how to share information, such that local data can be kept, but the relationship between such data across organizations is known. Secondly, and based on that, we want to use our knowledge to influence the common set of "core components" [The10] for

supporting collaboration through the Internet, as we believe that emergency management is an especially challenging area, but which should still be seamlessly supported by such core components.

References

[BH10] Achim D. Brucker and Dieter Hutter. Information Flow in Disaster Management Systems. In *International Conference on Availability, Reliability and Security (ARES)*, pages 156–163. IEEE Computer Society, 2010.

[BP09] Achim D. Brucker and Helmut Petritsch. Extending Access Control Models with Break-glass. In Barbara Carminati and James Joshi, editors, *ACM symposium on access control models and technologies (SACMAT)*, pages 197–206. ACM Press, 2009.

[Bun10] Bundesministerium für Bildung und Forschung (BMBF). Research for Civil Security – Rescue and Protection of People. `http://www.bmbf.de/pub/research_for_civil_security_rescue_protection_people.pdf`, 2010. [Online; accessed 06-Feb-2011].

[CDSDS+07] Mifan Careem, Chamindra De Silva, Ravindra De Silva, Louiqa Raschid, and Sanjiva Weerawarana. Demonstration of Sahana: free and open source disaster management. In *Digital government research: bridging disciplines & domains*, pages 266–267. Digital Government Society of North America, 2007.

[DEN11] deNIS – deutsche Notfallvorsorge-Informationssystem. `http://www.denis.bund.de/`, 2011. [Online; accessed 06-Feb-2011].

[IGN06] Berliner Feuerwehr-Leitstelle. `http://www.berliner-feuerwehr.de/leitstelle.html`, 2006. [Online; accessed 06-Feb-2011].

[PKA+10] Apostolos Papageorgiou, Tronje Krop, Sebastian Ahlfeld, Stefan Schulte, Julian Eckert, and Ralf Steinmetz. Enhancing Availability with Self-Organization Extensions in a SOA Platform. In *Fifth International Conference onInternet and Web Applications and Services (ICIW)*, pages 161 –166, 2010.

[PP10] Heiko Paulheim and Florian Probst. Application Integration on the User Interface Level: an Ontology-Based Approach. *Data & Knowledge Engineering Journal*, 69(11):1103–1116, 2010.

[TCdWY04] Murray Turoff, Michael Chumer, Bartel Van de Walle, and Xiang Yao. The Design of Emergency Response Management Information Systems (ERMIS). *The Journal of Information Technology Theory and Application (JITTA)*, 5:1–35, 2004.

[The10] The European Future Internet Initiative (EFII). Position paper on the Core Platform needed to support the Future Internet Projects. `http://www.future-internet.eu/fileadmin/news/EFII_Draft_Discussion_Paper-Architecture-May2010.pdf`, 2010. [Online; accessed 06-Feb-2011].

[TUV] TUV Rheinland. Industrial Risks Assessment and Analysis – DISMA. `http://www.tuv.com/ar/en/evaluacion_y_analisis_de_riesgos.html`.

[Und10] Sarah Underwood. Improving disaster management. *Commun. ACM*, 53:18–20, February 2010.



Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation (PIK2011)

Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation

PIK2011 Workshop-Einleitung und Überblick

Andreas Birk[1], Klaus Schmid[2], Markus Völter[3]

[1] Software.Process.Management, Usedomstraße 15, 70439 Stuttgart
andreas.birk@swpm.de

[2] Universität Hildesheim, Institut für Informatik
Marienburger Platz 22, 31141 Hildesheim
schmid@sse.uni-hildesheim.de

[3] voelter - ingenieurbüro für softwaretechnologie/itemis
Ötztaler Straße 38, 70327 Stuttgart
voelter@acm.org

Abstract: Produktlinien sind heute in vielen Bereichen der Software-Industrie vertreten. Dennoch bergen sie für viele Unternehmen noch bedeutende Herausforderungen und Risiken. Der Workshop beleuchtet aktuelle Erfahrungen mit Produktlinien und bietet ein Forum für den Dialog zwischen Praxis und anwendungsorientierter Forschung. Im Mittelpunkt steht das Wechselspiel zwischen technischen Fragestellungen und den geschäftlichen, organisatorischen und Prozessaspekten.

1 Produktlinien

Produktlinien sind heute in vielen Bereichen der Software-Industrie vertreten, von eingebetteten Systemen bis zu betrieblichen Informationssystemen. Sie ermöglichen höhere Produktivität, steigern die Qualität und verbessern die strategischen Positionen der Unternehmen, u.a. aufgrund eines hohen Grades an Wiederverwendung und Standardisierung.

Dennoch bergen Produktlinien für viele Unternehmen noch bedeutende Herausforderungen und Risiken. Die Gründe liegen teilweise im technischen Bereich. So sind viele Produktlinien-Technologien für den breiten Einsatz in der Praxis noch nicht genügend ausgereift und miteinander integriert. Die wohl größten Herausforderungen stellen sich in den Wechselwirkungen zwischen den technischen Verfahren mit den Prozessen sowie dem organisatorischen und geschäftlichen Kontext der Produktlinienentwicklung. —Wie müssen die technologischen Ansätze auf diese Wechselwirkungen ausgerichtet sein? Welche Potenziale bieten neue technologische Entwicklungen in unterschiedlichen Einsatzfeldern?

Der Workshop „Produktlinien im Kontext“ [PIK11] will aktuelle Erfahrungen mit Produktlinien beleuchten und den Dialog zwischen Praxis und anwendungsorientierter Forschung fördern. Im Mittelpunkt steht das Wechselspiel zwischen technischen Fragestellungen und den geschäftlichen, organisatorischen und Prozessaspekten. Daneben sollen auch neue technologische Entwicklungen vorgestellt und diskutiert werden.

2 Themengebiete

Die angenommenen Workshop-Beiträge behandeln die folgenden Themengebiete:

- Modellgetriebene Produktlinien
- Software-Architektur für Produktlinien
- Requirements-Management für Produktlinien

Der Workshop beinhaltet einen ausführlichen Diskussionsteil sowie ein Mini-Tutorial. Die Diskussionen sind offen für Themen, die sich während des Workshops ergeben. Zusätzlich zu den oben genannten Themen betrachten die Diskussionen und das Mini-Tutorial zumindest die folgenden Aspekte:

- Testen von Software-Produktlinien
- Der Lebenszyklus von Software-Produktlinien und die wesentlichen Herausforderungen in den einzelnen Lebenszyklus-Phasen
- Die Rolle von Software-Tools in der Produktlinienentwicklung

3 Beiträge und Diskussionen

Dieses Kapitel stellt die Workshop-Beiträge kurz vor und präsentiert die vorgesehenen Diskussionsblöcke. Die Diskussionen werden nach dem Workshop zusammengefasst und auf der Workshop-Homepage veröffentlicht [PIK11].

Der Beitrag von Buchmann und Dotor [BD11] stellt eine modellgetriebene Produktlinie für Softwarekonfigurationsverwaltungssysteme vor. Sie dient als Forschungsinstrument zur vergleichenden Untersuchung unterschiedlicher Verfahren zur Konfigurationsverwaltung. Zugleich bietet sie ein gut nachvollziehbares Beispiel für die Entwicklung größerer Software-Systeme aus Produktlinien. Insbesondere liefert das Projekt Erfahrungen zur Feature-Modellierung sowie zur modellgetriebenen Systementwicklung.

Frenzel, Friebe, Giesecke und Luhmann [Fre11] berichten Erfahrungen aus der Entwicklung einer industriellen Produktlinie für Systeme zum Energie-Prozess-Management in den Bereichen der Energieversorgung und Kraftwerkstechnik. Es handelt sich dabei um eine hierarchische Produktlinie, bei der Produktbestandteile wiederum Instanzen einer Produktlinie sind. Variationspunkte werden über Standardschnittstellen umgesetzt, hinter denen unterschiedliche, variantenspezifische Implementierungen stehen. Der Erfahrungsbericht betrachtet insbesondere Herausforderungen beim Einsatz von Standardschnittstellen sowie Aspekte des Testens.

Die automatische Verifikation von Produktlinien-Requirements behandelt der Beitrag von Markert und Oster [MO11]. Er ist angesiedelt im Bereich der eingebetteten Systeme und bezieht sich mit einem Anwendungsbeispiel auf die Steuerung von Autositzen. Ausgangspunkt ist ein Verfahren aus der Hardware-Entwicklung. Die Autoren haben es für die Anwendung auf Software-Produktlinien erweitert. Das Verfahren erlaubt es, bei sich ändernden Spezifikationen die Konsistenz der Requirements zu überwachen.

Oster [Ost11] gibt in einem Mini-Tutorial einen systematischen Überblick über Verfahren zum Testen von Produktlinien, speziell mit modellbasierten Methoden. Er stellt die Testverfahren einander gegenüber anhand von Kriterien wie Anwendbarkeit im industriellen Umfeld, Potenzial zur Wiederverwendung von Testfällen und erreichbare Testabdeckungsgrade. Der Autor schlägt weitere Verbesserungen der Verfahren vor und berichtet über Erfahrungen.

Die Diskussion des PIK 2011 Workshop wird eingeleitet durch einen Überblick über den Lebenszyklus von Software-Produktlinien [BSV11]. Er wird moderiert von den Organisatoren des Workshops. Da Herausforderungen bei der Produktlinienentwicklung sich in der Regel in einer speziellen Lebenszyklusphase manifestieren, bietet diese Betrachtung eine gute Grundlage zum Verständnis und zur Lösung der Herausforderungen. Kurzvorträge zu industriellen Fallbeispielen dienen als Einstieg in die Diskussion. Sie erleichtern es den Workshop-Teilnehmern, eigene Erfahrungen und Fragestellungen beizutragen.

Die Diskussions-Session soll insbesondere von den aktuellen Interessen und Anliegen der Workshop-Teilnehmer geprägt sein. Außerdem werden hier auch die relevanten Diskussionsergebnisse der zurückliegenden PIK-Tagungen eingebracht [BFS08][Bi09][BSV10]. Ein abschließender Programmteil fasst die Diskussionen zusammen, identifiziert die Kernpunkte und zeigt Querbeziehungen zwischen den Teildiskussionen auf. Die Ergebnisse der Diskussion werden auf der PIK-Webseite veröffentlicht [PIK11].

Literaturverzeichnis

[BD11] Buchmann, T.; Dotor, A.: MOD2-SCM: Eine modellgetriebene Produktlinie für Softwarekonfigurationsverwaltungssysteme. In: Software Engineering 2011 (Workshopband). Lecture Notes in Informatics (LNI), Bonn, Germany, 2011.

[BFS08] Birk, A.; Fiege, L.; Schmid, K.: Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation. PIK2008 Workshop-Einleitung und Überblick. In: Software Engineering 2008 (Workshopband). Lecture Notes in Informatics (LNI), P-122, Bonn, Germany, 2008.

[Bi09] Birk, A.; Fiege, L.; Kolagari, R.T.; Schmid, K.: Tagungsband der PIK 2009: Produktlinien im Kontext. Universität Hildesheim, Institut für Informatik, Hildesheim, Germany, 2009.

[BSV10] Birk, A.; Schmid, K.; Völter, M.: Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation. PIK2010 Workshop-Einleitung und Überblick. In: Software Engineering 2010 (Workshopband). Lecture Notes in Informatics (LNI), P-160, Bonn, Germany, 2010.

[BSV11] Birk, A.; Schmid, K.; Völter, M.: Der Lebenszyklus variantenreicher Produkte: Organisation, Prozesse, Tools. In: Software Engineering 2011 (Workshopband). Lecture Notes in Informatics (LNI), Bonn, Germany, 2011

[Fre11] Frenzel, M.; Friebe, J.; Giesecke, S.; Luhmann, T.: Standardschnittstellen als nichtfunktionale Variationspunkte: Erfahrungen aus der EPM-Produktlinie. In: Software Engineering 2011 (Workshopband). Lecture Notes in Informatics (LNI), Bonn, Germany, 2011.

[MO11] Markert, F.; Oster, S.: A Formal Method to Identify Deficiencies of Functional Requirements for Product Lines of Embedded Systems. In: Software Engineering 2011 (Workshopband). Lecture Notes in Informatics (LNI), Bonn, Germany, 2011.

[Ost11] Oster, S.: State-of-the-Art in Software Product Line Testing and Lessons Learned. In: Software Engineering 2011 (Workshopband). Lecture Notes in Informatics (LNI), Bonn, Germany, 2011.

[PIK11] PIK 2011 - Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation. URL: http://2011.pik-konf.de/

PIK 2011 Workshop-Organisation

Organisatoren

Andreas Birk, Software.Process.Management

Klaus Schmid, Universität Hildesheim

Markus Völter, Völter Ingenieurbüro und itemis

Programmkomitee

Danilo Beuche, pure-systems

Ludger Fiege, Siemens

Samuel Fricker,
Blekinge Institute of Technology

Paul Grünbacher, Universität Linz

Gerald Heller,
Software Process Consulting

Georg Herzwurm, Universität Stuttgart

Frank Houdek, Daimler

Dirk Janzen, Harman/Becker

Olaf Kath, ikv++

Michael Kircher, Siemens

Andreas Kreß, HOOD-Group

Kim Lauenroth,
Universität Duisburg-Essen

Klaus Marquardt, Dräger Medical

Thomas von der Maßen,
Arvato/Bertelsmann

Sebastian Oster, TU Darmstadt

Klaus Pohl, Universität Duisburg-Essen

Mark-Oliver Reiser, TU Berlin

Dietmar Schütz, Siemens

Reinhard Stoiber, Universität Zürich

Ramin Tavakoli, Volvo

Karina Villela, Fraunhofer IESE

Matthias Weber, Carmeq

MOD2-SCM: Eine modellgetriebene Produktlinie für Softwarekonfigurationsverwaltungssyteme

Thomas Buchmann, Alexander Dotor

vorname.nachname@uni-bayreuth.de

Abstract: Heutige Softwarekonfigurationsverwaltungssysteme (SCMS) sind monolithische und schwer erweiterbare Systeme, die mit großem Aufwand an die modernen Entwicklungsprozesse angepasst oder sogar extra neu implementiert werden. Daher werden sie unsystematisch modifiziert und neu entwickelt. So ist es nahezu unmöglich, systematische Vergleiche zwischen bestehenden SCMS anzustellen. Zusätzlich sind die meisten SCM Verfahren nur implizit durch den Quellcode beschrieben, bzw. bei proprietären Systemen nur aus Black-Box-Tests ableitbar. Dies führt letztendlich dazu, dass die Methoden der SCM-Domäne nur aufwändig verglichen und erforscht werden können. Insbesondere die Abhängigkeiten unterschiedlicher Verfahren werden durch die enge Kopplung der monolithischen Systeme verschleiert. In diesem Papier beschreiben wir die Entwicklung einer modellgetriebenen Produktlinie für Softwarekonfigurationsverwaltungssysteme. Es wird ein allgemeines Featuremodell für SCM Systeme beschrieben und ein Prototyp für eine neue Generation von SCM Systemen vorgestellt. Durch die Beschreibung der Produktlinie in Modellen wird außerdem der Abstraktionsgrad erhöht und der Zugang zu existierenden Methoden durch die Verwendung graphischer Modelle erleichtert.

1 Einleitung

Es existieren zur Zeit über 70 verschiedene Softwarekonfigurationsverwaltungssysteme (kurz: SCMS), um die Softwareentwicklung unterstützen. SCMS sind selbst auch Softwaresysteme, deren Größe einige zehntausend bis zu mehreren Millionen Quelltext-Zeilen umfasst. Trotz dieses Umfangs handelt es sich meist um monolithische Systeme, deren Daten und Funktionen eng verschränkt sind und deren Verfahren lediglich implizit durch den Quellcode beschrieben werden. Eine Identifikation einzelner Methoden ist kaum möglich, ebensowenig wie eine Wiederverwendung bereits implementierter Verfahren. Dies führt dazu, dass für ein neues Verfahren, oder sogar nur für eine neue Kombination existierender Verfahren, ein SCMS von Grund auf neu implementiert wird. Die fehlende Modularität erschwert auch die Wiederverwendung eines SCMS für unterschiedliche Software-Entwicklungsprozesse, da bereits kleine Unterschiede zwischen den Prozessen die Entwicklung eines neuen SCMS erfordern können. Es besteht daher der Bedarf nach einer neuen Generation von SCMS, die durch ihre modulare Architektur (1) die Wiederverwendung bestehender Verfahren erlauben, (2) die Anpassbarkeit an Entwicklungsprozesse verbessern bzw. überhaupt erst ermöglichen und (3) die Erweiterbarkeit verbessern, um ein SCMS in möglichst vielen Entwicklungsprozessen einsetzen zu können.

Die modellgetriebene Modulare Produktlinie für Softwarekonfigurationsverwaltungssysteme (kurz: MOD2-SCM) [Dot11, BD09c] identifiziert Gemeinsamkeiten von SCMS auf Basis eines Entwicklungsprozesses für modellgetriebene Software Produktlinien (MOD-PL) [BDW10, Buc10]. Der Entwicklungsaufwand eines SCMS wird so reduziert, da gemeinsame Komponenten wiederverwendet und die Produkte aus dem Domänenmodell generiert werden (anstatt sie von Hand zu implementieren). So kann durch Konfiguration des Domänenmodells nach den Anforderungen eines Softwareentwicklungsprozesses ein SCMS speziell an den zu unterstützenden Prozess angepasst werden.

Im Folgenden geben wir einen Überblick über das verwendete Featuremodell für Softwarekonfigurationsverwaltungssysteme sowie das generisches Domänenmodell, das auf diesem Featuremodell aufbaut und die Grundlage für die Systemfamilie bildet. Das Featuremodell erlaubt nicht nur die systematische Beschreibung der generierten SCMS, sondern auch einen Vergleich und die Klassifikation bereits bestehender monolithischer Systeme. Durch das MOD2-SCM-Domänenmodell wird die SCM-Domäne mit Hilfe von Komponenten und Klassen beschrieben. Abhängigkeiten und Kopplungen werden dabei explizit identifiziert und reduziert. Dies führt zu einer vertieften Einsicht in die SCM-Domäne speziell hinsichtlich der Modularisierbarkeit, Erweiterbarkeit und Co-Evolution [BDW08b]. Durch die Verwendung von Storydiagrammen [ZSW99] zur Verhaltensmodellierung kann aus dem Domänenmodell ausführbarer Quelltext generiert werden.

2 Das MOD2-SCM Featuremodell

2.1 Domänenanalyse

Zur Analyse der SCM-Domäne wurde FORM [KKL+98] - eine Weiterentwicklung von FODA [KCH+90] verwendet. FORM unterscheidet vier Ebenen:

Capability Layer: Features auf dieser Ebene beschreiben die Produktlinie durch Dienste, Operationen und Qualitäten, die angeboten werden sollen.

Operating Environment Layer: Die Features, die hier spezifiziert werden, beschreiben die Produktlinie durch Eigenschaften der Hard- und Softwareumgebungen, in denen die konkreten Anwendungen später eingesetzt werden.

Domain Technology Layer: Die Techniken, die speziell in der Anwendungsdomäne verwendet werden, werden mit Features auf dieser Ebene beschrieben. In der MOD2-SCM Produktlinie werden hier etwa die Features zur Realisierung der unterschiedlichen Module (Produktraum, Versionsraum, etc.) definiert.

Implementation Technique Layer: Auf dieser Ebene beschreiben Features die Produktlinie mit Hilfe der verwendeten grundlegenden Techniken. Dabei handelt es sich nicht um domänenspezifische Techniken, sondern um Basistechniken der Datenübertragung und -verarbeitung.

Das Ergenis der Domänenanalyse nach FORM wird in einem Featuremodell festgehalten.

2.2 Featuremodell

Das Featuremodell wurde mit Hilfe des Werkzeugs *FeaturePlugin* [AC04] erstellt und umfasst 147 Features, sowie 34 aussagenlogische Einschränkungen. Dabei ist zu beachten, dass nicht alle Features zur Auszeichnung von Modellelementen verwendet werden. 45 Features sind Erweiterungspunkte, die lediglich zur Klassifikation dienen, und nicht implementiert wurden. Von den verbliebenen 102 Features sind jedoch nur 45 Merkmale für Featureannotationen verwendet worden. Dies hat mehrere Gründe: Erstens beeinflussen einige Merkmale über die aussagenlogischen Einschränkungen die Konfiguration. Zweitens dienen verpflichtende Merkmale zur Strukturierung der variablen Merkmale, wurden aber nicht zum Markieren verwendet. In MOD2-SCM dient das Featuremodell sowohl zur Klassifikation von SCMS als auch zur Auszeichnung des Domänenmodells (vgl. Abschnitt 3.2).

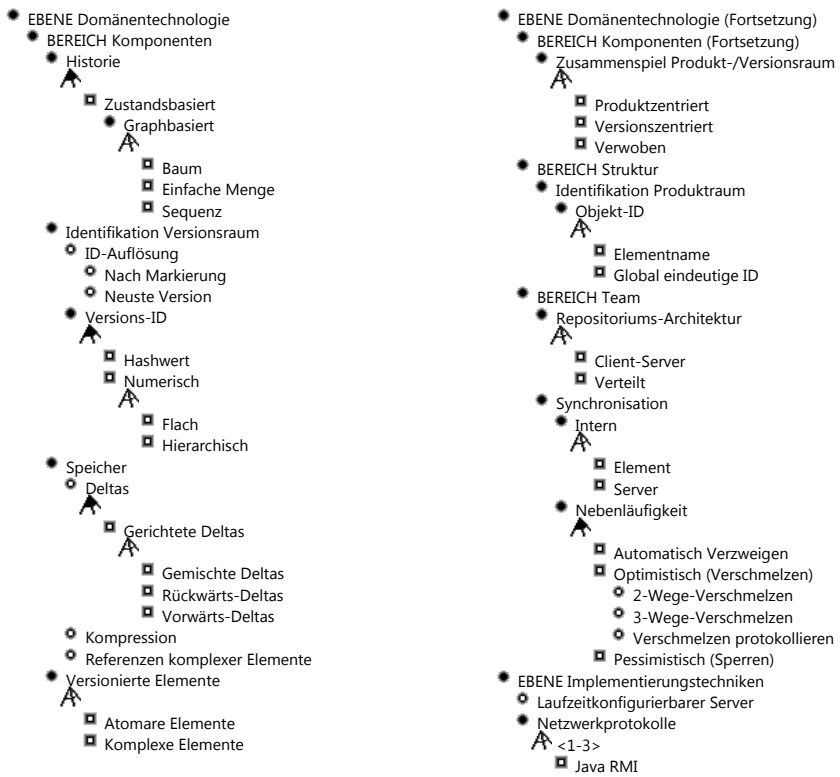


Abbildung 1: Ausschnitt aus dem Featuremodell der MOD2-SCM Produktlinie.

Der in Abbildung 1 gezeigte Ausschnitt des Featuremodells zeigt einen Teil der Ebene *Domain Technology Layer* (EBENE Domänentechnologie in der Abbildung). Diese Ebene enthält einen Bereich mit sämtlichen Komponenten, die für ein SCM-System benötigt werden: Die *Historie* zum Speichern der Versionen und ihrer Abhängigkeiten. Die *Identifikati-*

on [der Versionen im] Versionsraum, sowie zusätzliche *Meta-Informationen* zur Identifikation. Der *Speicher*, der die Versionen - meist platzsparend - persistenziert. Die Komplexität der *Versionierten Elemente* und die Möglichkeit parallel Entwicklungen durch *Verzweigen* zu unterstützen, sowie letztendlich das *Zusammenspiel von Produkt-/Versionsraum*. Im Bereich *Struktur* ist dies die *Identifikation [der Elemente im] Produktraum* und im Bereich *Team*: Die „Arbeitsbereich-Informationen" bestimmen, welche Informationen die SCMS-Operationen im Arbeitsbereich benötigen. Die *Repositoriums-Architektur* bestimmt den Aufbau und Kommunikationswege des SCMS, während die *Synchronisation* Verfahren zur Koordination mehrerer Entwickler beschreibt.

Aufgrund der erheblichen Größe des Featuremodells für MOD2-SCM kann im Rahmen dieses Aufsatzes nur ein kleiner Teil gezeigt werden. Das vollständige Featuremodell und die zugehörige Domänenanalyse ist in [Dot11] nachzulesen. Die Umsetzung dieser Features erfolgt im Domänenmodell.

3 Das ausführbare MOD2-SCM Domänenmodell

Das MOD2-SCM Domänenmodell beschreibt die Klassenstruktur bzw. das Verhalten der einzelnen Methoden mit Hilfe von Klassen- bzw. Storydiagrammen in Fujaba [Zün02]. Storydiagramme [ZSW99] dienen in Fujaba zur Modellierung des Verhaltens von genau einer Methode einer Klasse. Sie ähneln Interaktionsübersichtsdiagrammen in UML2 [OMG09]. Fujaba ist in der Lage aus Storydiagrammen ausführbahren Quelltext zu generieren, und somit modellgetriebenes Arbeiten zu unterstützen. Das Fujaba Modell wird durch ein Paketmodell aus dem MODPL-Paketdiagrammeditor ergänzt, das die grundlegende Architektur der MOD2-SCM Produktlinie mit Hilfe von Paketdiagrammen beschreibt. Dabei besteht eine 1:1-Verknüpfung zwischen den Paketen des MODPL-Paketmodells und des Fujaba-Modells (über den voll qualifizierten Namen). Zusätzlich ist auch jedes Klassendiagramm aus dem Fujaba-Modell genau einem Paket zugeordnet, dessen Inhalt es darstellt. Die Synchronisation der beiden Modelle erfolgt mittels des MODPL-Werkzeugkastens [Buc10, BDW10].

3.1 Architektur

3.1.1 Paketdiagramm

Bei der Erstellung des MOD2-SCM Paketmodells spielte auch die Variabilität und insbesondere eine ökonomische Auszeichnung mittels Featureannotationen eine Rolle. Daher wurden die Server-, Kommunikations- und Arbeitsbereich-Pakete ihren jeweiligen Modulen zugeordnet (anstatt z.B. drei Pakete für Server, Arbeitsbereich und Kommunikation zu wählen und die Komponenten darauf zu verteilen). Nur so ließ sich mit genau einer Featureannotation am entsprechenden Unterpaket das Domänenmodell konfigurieren, z.B. im Falle der Historie mit genau einem Feature am entsprechenden *history*-Unterpaket (vgl.

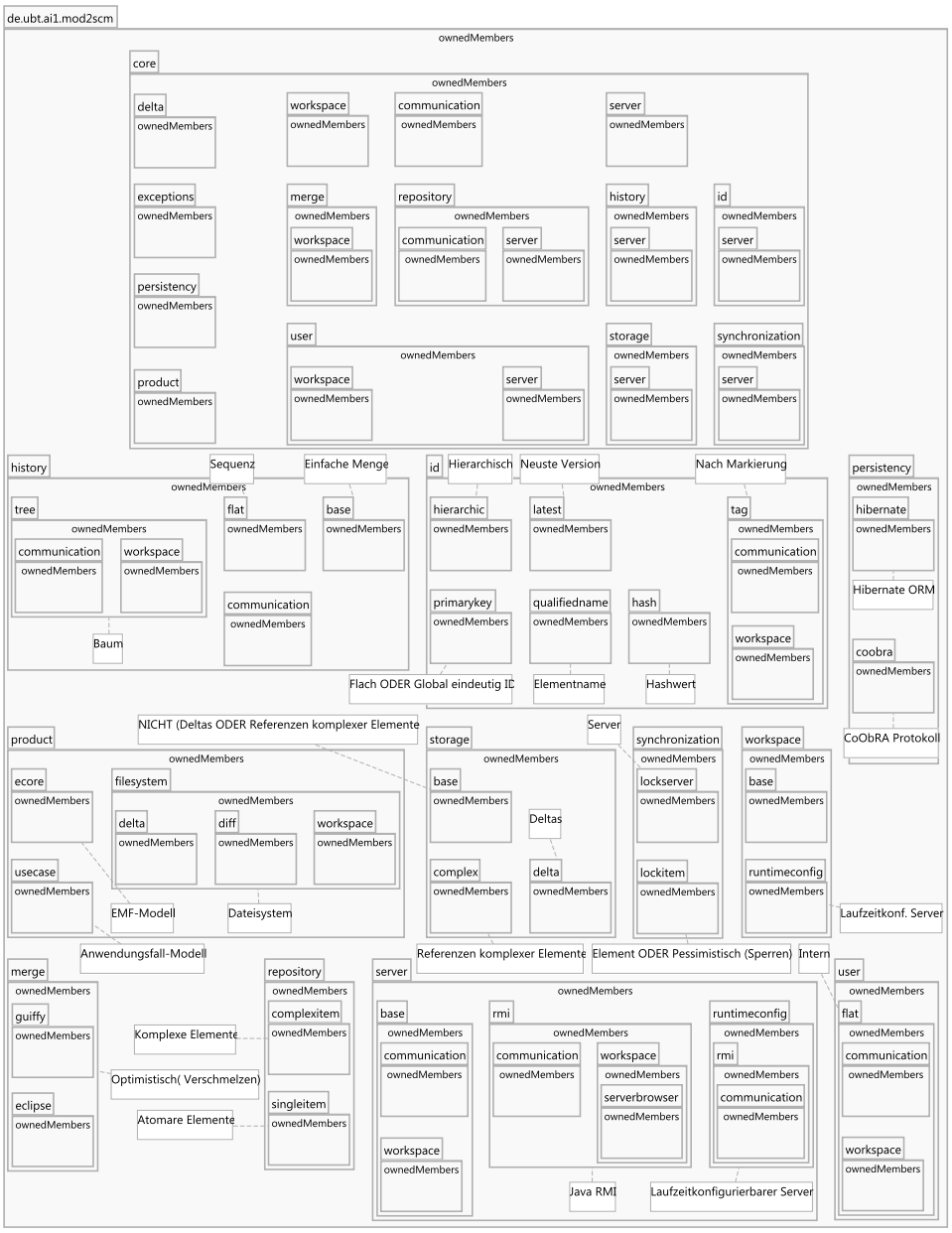


Abbildung 2: Das Paketdiagramm des MOD2-SCM Domänenmodells mit Featureannotationen.

Abbildung 2).

Abbildung 2 zeigt die Architektur des MOD2-SCM Domänenmodells. Es besteht aus 14 Kernmodulen und 29 Modulen der Modulbibliothek, die auf diesen Kernmodulen aufbau-

en. Einige Pakete sind hier bereits mit Featureannotationen ausgezeichnet. Bei der Auszeichnung werden die eindeutigen Bezeichner der jeweiligen Features aus dem Featuremodell und nicht der im Featuremodell angezeigte Name verwendet. Das von uns entwickelte Werkzeug stellt hierüber die Verknüpfung von Featuremodell und Domänenmodell her [BD09b, BDW10, Buc10]. Die Vergabe von Featureannotationen erfolgt durch Auswahl der jeweiligen Features im Featuremodell. Aufgrund von Platzrestriktionen können leider im Rahmen dieses Artikels keine weiteren Auszüge aus dem Domänenmodell (z.B. Klassen- oder Storydiagramme) präsentiert werden. Weiterführende Informationen zum Domänenmodell sind aber in [Dot11] nachzulesen.

3.1.2 Kontrolle der Kopplung

Für die Entwicklung einer Produktlinie für SCM-Systeme ist es essentiell wichtig, die Module, welche die unterschiedlichen Aufgabenbereiche eines SCM Systems realisieren, weitestgehend voneinander zu entkoppeln. Nur so ist es möglich, z.B. unterschiedliche Arten von Delta- und Versionsspeicherung orthogonal miteinander zu kombinieren. Daher müssen Abhängigkeiten zwischen solchen Modulen im Domänenmodell erfasst und womöglich beseitigt werden. Der von uns entwickelte Paketdiagrammeditor zur Unterstützung des *Modellieren im Großen* [BDK09, BDW10, Buc10] bietet hierfür eine Hilfestellung. Existierende Metriken zur Messung der Kopplung basieren i.d.R. auf Klassen und ihren Abhängigkeiten [BDW98]. Die Abhängigkeiten im MOD2-SCM Domänenmodell sind jedoch anhand der Pakete und ihren privaten Importbeziehungen beschrieben. Daher können sie analog zu den Abhängigkeiten definiert werden, die für die Existenz privater Paketimport-Beziehung zuständig sind, da sich diese unmittelbar auf die im Paket enthaltenen Klassen zurückführen lassen [Buc10]: Eine Abhängigkeit einer Klasse A von einer Klasse B, liegt genau dann vor, wenn (1) B ein Attributtyp von A ist, (2) B ein Rückgabetyp einer Methode von A ist (einschl. Ausnahmen), (3) B ein Parametertyp einer Methode von A ist, (4) B direkte Oberklasse von A ist, (5) B als Objekttyp in einem Storydiagramm von A verwendet wird. Die **Kopplungsstärke** $k(A;B)$ zwischen einer Klasse A und einer Klasse B lässt sich wie folgt berechnen:

$$
\begin{aligned}
k(A,B) &= |B\ Attributtyp\ in\ A| \\
&+ |B\ Rückgabetyp\ in\ A| \\
&+ |B\ Parametertyp\ in\ A| \\
&+ |B\ Oberklasse\ von\ A| \\
&+ |B\ Objekttyp\ in\ A|
\end{aligned}
$$

Abbildung 3 zeigt einen Ausschnitt aus dem Paketdiagramm des MOD2-SCM Domänenmodells und die unterschiedlich starke Kopplung zwischen den einzelnen Paketen. Die Liniendicke gibt dabei den Grad der Kopplung an. Hierbei kommen im graphischen Editor verschiedene Intervalle für die Darstellung der Liniendicke zum Einsatz. Dabei werden zunächst die Kopplungwerte aller Elemente in den jeweiligen Paketen ermittelt und summiert: Ist die Summe dann 1, so beträgt die Liniendicke ebenfalls 1. Die Liniendicke ist

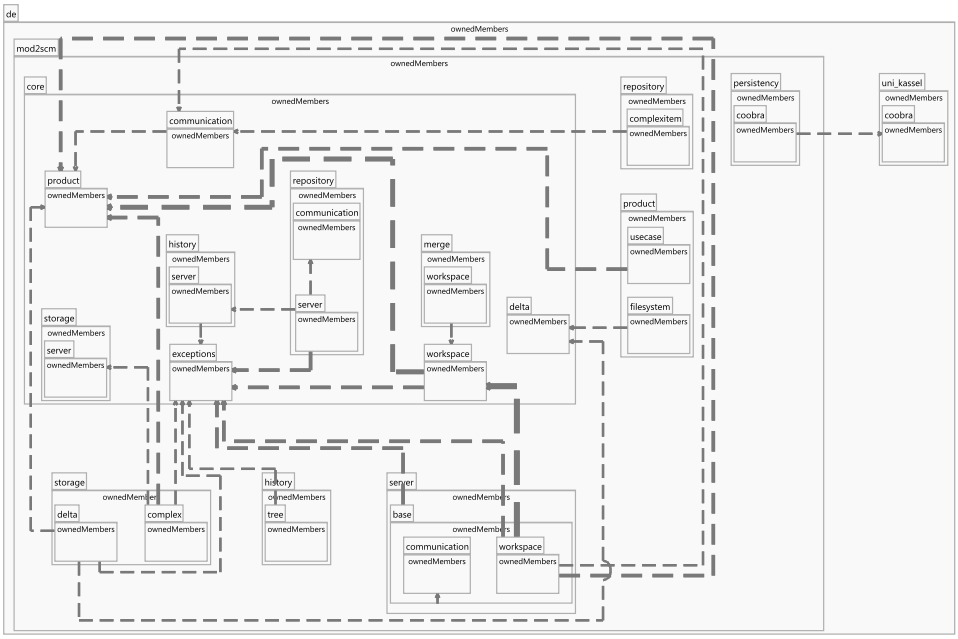


Abbildung 3: Grad der Kopplung zwischen den Paketen wird durch unterschiedlich dicke Linien ausgedrückt.

2, wenn die Summe Werte zwischen 2 und 9 annimmt, 4 bei Werten zwischen 10 und 19, usw. Das Maximum der Liniendicke beträgt 10 und gilt für Werte > 40.

Das vollständige MOD2-SCM Domänenmodell besteht aus 14 Kernmodulen und 29 Modulen in der Modulbibliothek (siehe auch Abb. 2). Zwischen ihnen bestehen 244 Abhängigkeiten, größtenteils zwischen Kernmodulen bzw. einem austauschbaren Modul zu seinem Kernmodul. Die Austauschbarkeit wird vor allem durch Schnittstellen-Abstraktion erreicht, so dass jedes Bibliotheksmodul von der Schnittstelle eines Kernmoduls abhängt. Es existieren kaum Kopplungen unter den Bibliotheksmodulen, somit sind diese gegen beliebige Module mit der gleichen Schnittstelle austauschbar. Anhand der Kopplungsanzahl und -stärke ließen sich auffällige Module identifzieren und genauer untersuchen. Da jedes Kernmodul einen Aufgabenbereich eines SCMS repräsentiert, lassen diese Kopplungen auch Rückschlüsse auf die Abhängigkeiten der SCM-Konzepte zu. Eine quantitative Analyse der Kopplungen liefert schnell gute Indikatoren für „kritische“ bzw. „interessante“ Pakete, lässt sich aber nur schwer zur Bewertung der Architekturqualität einsetzen. Zwischen den Modulen der Modulbibliothek existieren jedoch nur wenige Abhängigkeiten, so dass sich fast alle Module beliebig austauschen lassen. Die Entkopplung der einzelnen Module innerhalb der MOD2-SCM Produktlinie ist gelungen.

3.2 Verbindung mit dem Featuremodell

Die Verbindung zwischen dem Featuremodell und dem Domänenmodell wurde mit Hilfe des Werkzeugs *MODPLFeaturePlugin* [BD09a, BD09b, BDW10, Buc10] hergestellt. Hierzu wurden die entsprechenden Modellelemente mit den jeweiligen Features annotiert. Der Teil des Featuremodells, der mit dem Domänenmodell realisiert wurde, umfasst insgesamt 147 Features. Letztendlich war es aber nur erforderlich 57 Elemente des Domänenmodells manuell mit Features auszuzeichnen. Die Featureannotationen verteilen sich dabei wie folgt:

- 20 Pakete
- 8 Klassen
- 7 Methoden
- 22 Aktivitäten in Storymustern

Nur diese Modellelemente wurden manuell mit Features ausgezeichnet. Der MODPL-Werkzeugkasten unterstützt den Anwender durch automatische Propagation und Reparaturaktionen von Featureannotationen. Durch diesen Mechanismus wird sichergestellt, dass das konfigurierte Modell bzw. der konfigurierte Quelltext syntaktisch korrekt sind [BD09a, BD09b, BDW10, Buc10]. Hierbei werden Featureannotationen an abhängige Modellelemente automatisch übertragen, so dass bei der Konfigurierung keine Verletzungen des Metamodells entstehen.

Im MODPL-Werkzeugkasten bedeutet das Annotieren eines Modellelements, dass dieses nur dann in einem konfigurierten SCMS existiert, wenn das Feature Teil der Konfiguration ist [Buc10]. Soll ein Modellelement existieren, wenn ein Feature **nicht** in einer Konfiguration ist - z.B. als Alternative zu einem Modellelement mit ausgewähltem Feature - lässt sich dies nicht modellieren, da keine Annotation mit Negation existiert. Um diese Einschränkung zu umgehen, wurden Hilfsmerkmale im Featuremodell eingeführt, die über eine aussagenlogische Verknüpfung an das betroffene Feature gekoppelt sind, und immer den gegenteiligen Wert annehmen. Die von uns eingesetzte Software zur Featuremodellierung (*FeaturePlugin* [AC04]) unterstützt dies.

4 Vergleich mit anderen Ansätzen

4.1 Ähnliche Ansätze

4.1.1 Generische Versionsmodelle

Dieser Ansatz stellt keinen klassischen Produktlinienansatz dar, vielmehr wird hier versucht, ein generisches Versionsmodell zu entwickeln. Generische Versionsmodelle verwenden ein einheitliches Meta-Datenmodell, das als Schema für die Implementierung

unterschiedlicher Datenmodelle eingesetzt wird. Ziel dieser Methode ist es, mit einem einzigen Verfahren möglichst viele Datenmodelle mit Hilfe elementarer Datentypen zu beschreiben. Wiederverwendung wird in dieser Methode durch die Wiederverwendung der Elementaroperationen erreicht. Prominente Vertreter dieses Ansatzes sind ECM/UVM [WMC01] und ICE [Zel95, ZS97].

4.1.2 Bamboo

Bamboo [WGP04] ist ein Rahmenwerk, bei dem die Funktionalität eines SCMS manuell implementiert und anschließend in Quelltextfragmente zerlegt wurde. Jedes dieser Fragmente wird nun einem oder mehreren SCMS zugeordnet, in denen es verwendet wird. Durch Auswahl einer bestimmten Implementierung, werden die benötigten Quellcode-Fragmente selektiert und zu einem SCMS zusammengesetzt. Wiederverwendung wird in dieser Methode durch Wiederverwendung von Quellcode-Fragmenten in mehreren manuellen Implementierungen erreicht. Doch obwohl Vorlagen verwendet werden, handelt es sich nicht um eine dynamische, vorlagenbasierte Quellcode-Generierung, da für jede Variante die Quellcode-Fragmente explizit vorliegen müssen [Guo08]. Bamboo verwendet sogenannte Containment Models, eine spezialisierten Fassung von Entity/Relationship Diagrammen, zur Beschreibung des Datenmodells [Whi02, Whi03].

4.1.3 VS-Gen

Quellcode-Generatoren erzeugen Quellcode aus grafischen Modellen, indem sie die Modellelemente auf Quellcode-Vorlagen abbilden. Durch Annotation der Modellelemente kann die Abbildung zusätzlich noch parameterisiert werden, um die Quellcode-Erzeugung zu steuern. Meist lässt sich jedoch nur der Quellcode für die Struktur und nicht für das Verhalten generieren, so dass das Verhalten mit Hilfe der Quellcode-Vorlagen spezifiziert wird. Mit Hilfe der Annotationen können dann die benötigten Quellcode-Fragmente in der Vorlage gewählt werden, um so - ähnlich wie bei der bedingten Übersetzung - das gewünschte Verhalten zu erzeugen. Wiederverwendung wird in dieser Methode durch die Wiederverwendung der Quellcode-Vorlagen beim Generieren eines SCMS erreicht. Durch Annotation der Modellelemente mit den Merkmalen einer Produktlinie ist diese Methode ein Schritt in Richtung modellgetriebener Produktlinien. VS-Gen [KG04, Kov05] stellt einen solchen Ansatz dar.

4.2 Bewertung

In den vergangenen Jahren, gab es einige unterschiedliche Ansätze, eine Produktlinie für Versionskontrollsysteme zu erstellen. Allerdings beruhen viele davon auf manueller Implementierung. In der Anwendungsentwicklung müssen die Quellcodefragmente händisch zu einem lauffähigen System kombiniert werden, was mit einem oftmals nicht unerheblichen Aufwand verbunden ist. In einigen Ansätzen wird auf die Verwendung von formalen Modellen komplett verzichtet, andere benutzen sie lediglich zur Dokumentation bzw. zur

Beschreibung der statischen Struktur. In dem hier vorgestellten Ansatz hingegen werden sowohl Struktur als auch Verhalten der Produktlinie für SCM-Systeme durch formale Modelle beschrieben. Da das Domänenmodell alle Varianten umfasst, reduziert sich somit die Anwendungsentwicklung auf einen reinen Konfigurierungsschritt, an dessen Ende vollautomatisch ausführbarer Quelltext erzeugt wird.

5 Zusammenfassung

Die MOD2-SCM Produktlinie ist eine modellgetrieben entwickelte komplexe Produktlinie. Sie zeigt, dass sich mit dem in [Buc10, BDW10] vorgestellten MODPL-Ansatz und -Werkzeugkasten Produktlinien modellgetrieben entwickeln lassen. Dieser Ansatz bietet mehrere Vorteile gegenüber herkömmlicher Produktlinien-Entwicklung: Größter Vorteil ist, dass das manuelle Ableiten bzw. Zusammensetzen eines Software-Produktes aus dem Domänenmodell durch eine automatisierte Konfiguration ersetzt wird. Dabei kann sowohl sofort lauffähiger Quellcode generiert werden als auch ein konfiguriertes Modell, um es mit produktspezifischen Komponenten zu erweitern [Buc10]. Weiterhin wurde das Domänenmodell vollständig modellgetrieben entwickelt.

Durch die explizite Verknüpfung der Modellelemente aus Featuremodell und Domänenmodells sind die Abhängigkeiten von Features und Modell-Elementen zurückverfolgbar. Und auch die Architektur der Produktlinie lässt sich anhand von Paketdiagrammen grafisch darstellen und ebenfalls explizit mit den Features in Verbindung bringen. Ebenso lässt sich die Konfiguration eines Produktes grafisch im Domänenmodell darstellen sowie verwendete und nicht verwendete Modellelemente parallel anzeigen [Buc10].

Im Verlauf des Projektes zeigte sich, dass durch umsichtige Nutzung von objektorientierten Modellierungstechniken und den bewussten Einsatz von Entwurfsmustern der Annotationsaufwand für den Benutzer äußerst gering gehalten werden kann. Obwohl die von uns entwickelten Werkzeuge zur Verbindung von Featuremodell und ausführbahrem Domänenmodell die Annotation von Modellelementen mit Features aus dem Featuremodell auf einer beliebigen Granularitätsstufe erlauben [Buc10, BDW10], reichte es in den meisten Fällen aus, Featureannotationen auf Paketebene zu vergeben [Dot11]. Auch zeigte sich, dass es meist ausreicht Modellelemente mit einzelnen Features zu annotieren. Komplexe Featureausdrücke waren nicht notwendig, da aussagenlogische Verknüpfungen bereits im Featuremodell vorgenommen werden können.

Weiterhin zeigte sich, dass sich einzelne Module eines Softwarekonfigurationsverwaltungssystems derart entkoppeln lassen, so dass aus den einzelnen Modulen eine Modellbibliothek als Grundlage einer Softwareproduktlinie erstellt werden kann. Um dies zu erreichen, war der von uns im Rahmen des Projektes entwickelte Paketdiagrammeditor [BDW08a, BDK09, BDW10, Buc10] sehr hilfreich, da er die Abhängigkeiten im Domänenmodell erfassen und visualisieren kann. Hierbei wurde ebenfalls deutlich, dass die in der UML Spezifikation [OMG09] definierten privaten Importbeziehungen zwischen Modellelementen am Besten für diese Zwecke geeignet sind, da sie, im Gegensatz zu den

öffentlichen Importbeziehungen, nicht transitiv arbeiten. Somit erreicht man durch die privaten Importe eine bessere Übersicht über erwünschte und unerwünschte Kopplungen im Domänenmodell.

Der aktuelle Stand der MOD2-SCM Produktlinie ermöglicht es, 290.304.000 verschiedene SCM Systeme zu generieren. Damit ist die Produktlinie so groß, dass sich nicht mehr alle Konfigurationen erzeugen und geschweige denn testen lassen. Lediglich mit Hilfe des laufzeitkongurierbaren Servers[1] können ca. 1.000 Kombinationen von Servermodulen getestet werden. Somit stellt das Testen ein noch offenes Forschungsproblem dar.

Als Benutzerschnittstelle für das MOD2-SCM System wurde ein Eclipse Plugin für eine spezielle Konfiguration mittels herkömmlicher Programmierung erstellt. Zukünftige Forschungsarbeiten könnten eine Ausdehnung der Produktlinie auf die Benutzerschnittstelle umfassen.

Literatur

[AC04] Michal Antkiewicz und Krzysztof Czarnecki. FeaturePlugin: feature modeling plug-in for Eclipse. In *eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, Seiten 67–72, New York, NY, USA, 2004.

[BD09a] Thomas Buchmann und Alexander Dotor. Constraints for a fine-grained mapping of feature models and executable domain models. Jgg. 1 of *CTIT Workshop Proceedings*, Seiten 9–17. CTIT, Juni 2009.

[BD09b] Thomas Buchmann und Alexander Dotor. Mapping Features to Domain Models in Fujaba. In Pieter van Gorp, Hrsg., *Proceedings of the 7th International Fujaba Days*, Seiten 20–24, Eindhoven, The Netherlands, November 2009.

[BD09c] Thomas Buchmann und Alexander Dotor. Towards a Model-Driven Product Line for SCM Systems. Jgg. 2, Seiten 174–181, San Francsisco, CA, USA, 2009. Software Engineering Institute, Carnegie Mellon University.

[BDK09] Thomas Buchmann, Alexander Dotor und Martin Klinke. Supporting Modeling in the Large in Fujaba. In Pieter van Gorp, Hrsg., *Proceedings of the 7th International Fujaba Days*, Seiten 59–63, Eindhoven, The Netherlands, November 2009.

[BDW98] Lionel C. Briand, John W. Daly und Jürgen Wüst. A Unified Framework for Cohesion Measurement in Object-OrientedSystems. *Empirical Softw. Engg.*, 3(1):65–117, 1998.

[BDW08a] Thomas Buchmann, Alexander Dotor und Bernhard Westfechtel. Experiences with Modeling in the Large in Fujaba. In Uwe Aßmann, Jendrik Johannes und Albert Zündorf, Hrsg., *Proceedings of the 6th International Fujaba Days*, Seiten 5–9, Dresden, Germany, 2008. Technische Universität Dresden.

[BDW08b] Thomas Buchmann, Alexander Dotor und Bernhard Westfechtel. MOD2-SCM: Experiences with co-evolving models when designing a modular SCM system. In Dirk Deridder, Jeff Gray, Alfonso Pierantonio und Pierre-Yves Schobbens, Hrsg., *1st International Workshop on Model Co-Evolution and Consistency Management (MCCM08)*, Jgg. 1, Seiten 50–65, Toulouse, France, September 2008.

[1] Der laufzeitkonfigurierbare Server erlaubt das Austauschen von einzelnen Modulen zur Laufzeit. In Verbindung mit JUnit4 können so parametrisierte Tests durchgeführt werden.

[BDW10] Thomas Buchmann, Alexander Dotor und Bernhard Westfechtel. Werkzeuge zur modellgetriebenen Entwicklung von Produktlinien: Ein Erfahrungsbericht am Beispiel von Versionskontrollsystemen. In Andreas Birk, Klaus Schmid und Markus Völter, Hrsg., *Tagungsband der PIK 2010 Produktlinien im Kontext*, 2010.

[Buc10] Thomas Buchmann. *Modelle und Werkzeuge für modellgetriebene Softwareproduktlinien am Beispiel von Softwarekonfigurationsverwaltungssystemen*. Doktorarbeit, Universität Bayreuth, September 2010.

[Dot11] Alexander Dotor. *Entwurf und Modellierung einer Produktlinie von Software-Konfigurations-Management-Systemen*. Doktorarbeit, Universität Bayreuth, Bayreuth, 2011. Submitted.

[Guo08] Ge Guozheng. *RHIZOME: A Feature Modeling and Generation Platform for Software Product Lines*. Dissertation, University of California, Santa Cruz, Dezember 2008.

[KCH+90] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak und A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Bericht, Carnegie-Mellon University Software Engineering Institute, November 1990.

[KG04] Jernej Kovse und Christian Gebauer. VS-Gen: A case study of a product line for versioning systems. In *Generative Programming and Component Engineering 2004, Proceedings*, Seiten 396–415, 2004.

[KKL+98] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Gerard Jounghyun Kim und Euiseob Shin. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[Kov05] J. Kovése. *Model-driven development of versioning systems*. Dissertation, Technische Universität Kaiserslautern, Kaiserslautern, 2005.

[OMG09] OMG. *OMG Unified Modeling Language (OMG UML), Superstructure V2.2*. OMG, 140 Kendrick Street, Building A, Suite 300 Needham, MA 02494 USA, Februar 2009. Version 2.2.

[WGP04] James E. Whitehead, GuoTheng Ge und Kai Pan. Automatic Generation of Version Control Systems. Bericht, University of California, 2004.

[Whi02] James E. Whitehead. Uniform comparison of data models using containment modeling. In *HYPERTEXT '02: Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, Seiten 182–191, New York, NY, USA, 2002.

[Whi03] James E. Whitehead. Uniform Comparison of Configuration Management Data Models. In *Software Configuration Management*, Seiten 70–85, 2003.

[WMC01] Bernhard Westfechtel, Björn P. Munch und Reidar Conradi. A Layered Architecture for Unfiorm Version Management. Jgg. 27, Seiten 1111–1133, Dezember 2001.

[Zel95] Andreas Zeller. A Unified Version Model for Configuration Management. In Gail Kaiser, Hrsg., *Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Jgg. 20 (4) of *ACM Software Engineering Notes*, Seiten 151–160. ACM Press, Oktober 1995.

[Zün02] Albert Zündorf. *Rigorous Object Oriented Software Development*, 2002.

[ZS97] Andreas Zeller und Gregor Snelting. Unified Versioning through Feature Logic. *ACM Transactions on Software Engineering and Methodology*, 6(4):397–440, Oktober 1997.

[ZSW99] Alber Zündorf, Andy Schürr und Andreas J. Winter. Story Driven Modeling. Technical Report tr-ri-99-211, University of Paderborn, 1999.

Standardschnittstellen als nichtfunktionale Variationspunkte: Erfahrungen aus der EPM-Produktlinie

Martin Frenzel, Jörg Friebe, Simon Giesecke, Till Luhmann

BTC Business Technology Consulting AG
Escherweg 5
26121 Oldenburg
{martin.frenzel, joerg.friebe, simon.giesecke, till.luhmann}@btc-ag.com

Abstract: Die Energie-Prozess-Management-Produktlinie (EPM-Produktlinie) der BTC AG baut auf dem gemeinsamen EPM-Architekturstil und der EPM-Referenzarchitektur auf. Deren Anwendung stellt die technische und fachliche Interoperabilität innerhalb der Produktlinie sicher, und ermöglicht es, die effiziente Wartbarkeit, Testbarkeit sowie die erforderliche Variabilität in den nichtfunktionalen Anforderungen und der technischen Umgebung in kundenspezifischen Anwendungslandschaften erzielen zu können. Zur Erreichung dieser Variabilität werden technologieübergreifende Standardschnittstellen als Variationspunkte eingesetzt, die von mehreren Komponenten mit unterschiedlichen nichtfunktionalen Eigenschaften bzw. Anforderungen an die technische Umgebung implementiert werden. Dieser Beitrag berichtet über Erfahrungen und Lösungsansätze für Herausforderungen, die sich in der EPM-Produktlinie gezeigt haben.

1 Einleitung

Die BTC AG bietet ein Portfolio von Softwareprodukten für das technisch-betriebliche Energie-Prozess-Management (EPM) an. Hierzu gehören beispielsweise Anwendungen in den Bereichen Leittechnik für Energieversorgungsnetze, Kraftwerke und Windparks, Advanced Meter Management und virtuelle Kraftwerke.

Auf dieser Ebene werden in verschiedenen Technologien umgesetzte Alt- und Neuprodukte in der EPM-Produktlinie zusammengeführt, um mit Mitteln der Produktlinienentwicklung neue Produkte effizienter und mit höherer Qualität bereitstellen zu können.

Die EPM-Produktlinie dient zunächst als Katalysator für die effektive Wiederverwendung innerhalb des EPM-Produktportfolios. Wiederverwendungsprogramme scheitern häufig an unzureichender technischer und fachlicher Interoperabilität der Softwarebausteine [GAO09]. Zentrale Mittel, um diese Effekte zu verhindern, sind ein gemeinsamer Architekturstil (EPM-Architekturstil), der übergreifende Architekturregeln festlegt, und eine Referenzarchitektur (EPM-Referenzarchitektur), die die funktionalen Zuständigkeiten der wiederverwendbaren Bausteine untereinander und gegenüber den verwendenden Produkten abgrenzt.

Komponentenorientierung und die Verwendung von Standardschnittstellen spielen eine wichtige Rolle innerhalb des EPM-Architekturstils, um die Austauschbarkeit von Komponenten zu erreichen. Diese dienen insbesondere als Variationspunkte hinsichtlich nicht-funktionaler Aspekte. Die Verwendung von Standardschnittstellen bietet Effizienzpotenziale, birgt aber auch Herausforderungen.

Im Folgenden wird zunächst ein Überblick über den Aufbau der EPM-Produktlinie und die Produktlinien-weiten Artefakte gegeben (Abschnitt 2). Anschließend wird ein Überblick über die Architektursichten gegeben, die für die Modellierung von Architekturen in der EPM-Produktlinie verwendet werden (Abschnitt 3). Abschnitt 4 beschreibt die Rolle von Standardschnittstellen in der EPM-Produktlinie. In Abschnitt 5 werden die bislang gemachten Erfahrungen und die dabei identifizierten Herausforderungen, insbesondere im Hinblick auf die Testbarkeit, beschrieben. Abschnitt 6 enthält einen Ausblick auf zukünftige Weiterentwicklungen des Ansatzes.

2 Die EPM-Produktlinie

2.1 Motivation

Innerhalb der EPM-Produktlinie gibt es einzelne Produkte, die teilweise wieder enger zusammenhänge Teilproduktlinien bilden. Die Produkte sind vorwiegend individualisierbare Standardprodukte, d.h. sie werden in Kundenprojekten durch die BTC AG auf die spezifischen Anforderungen des Kunden angepasst. Sie werden in unterschiedlichen Technologien, insbesondere C++ unter Windows und Linux und C#/.NET entwickelt.

Eine Optimierung von Entwicklungskosten, Lieferzeiten und Qualität der den Kunden der BTC angebotenen Produkte und Lösungen soll über die Produkt-übergreifende Wiederverwendung von Komponenten und Schnittstellen erzielt werden. Als Katalysator der Wiederverwendung wirken die Produktlinien-weiten Artefakte EPM-Referenzarchitektur und EPM-Architekturstil zusammen[1], die im Folgenden beschrieben werden.

2.2 EPM-Referenzarchitektur

Den Kern der EPM-Referenzarchitektur bildet eine Strukturierung, die sich in einem Zwiebelmodell mit fünf Schichten von Softwarebausteinen[2] darstellen lässt (siehe Abbildung 1). Von innen nach außen handelt es sich zunächst um (1) Basisdienste & Frameworks und (2) EPM-Plattformbausteine. Diese bilden gemeinsam die EPM

[1] Man könnte insgesamt im weiteren Sinne von der Produktlinienarchitektur sprechen, die aber oft im Zusammenhang mit Feature-Modellen betrachtet wird. Feature-Modelle mit funktionalen Variationspunkten spielen auf dieser Ebene eine geringere Rolle, dies ist eher bei den Teilproduktlinien zu finden.

[2] „Softwarebaustein“ wird hier als allgemeiner Oberbegriff für die je nach Architekturstandpunkte unterschiedlichen Modellierungselemente verwendet. Darunter fallen insbesondere Komponenten, Schnittstellen und Module (vgl. Abschnitt 3).

Universal Platform. Darauf folgen (3) Domänen-übergreifende Bausteine, die für mehrere Teilproduktlinien verwendet werden, und (4) Produktlinien-spezifische Bausteine, die jeweils nur für eine Teilproduktlinie verwendet werden. Die Schicht (5) ist geteilt in Produkt-spezifische Bausteine und Projekt-spezifische Bausteine, wobei letztere auf Produkt-spezifischen Bausteinen aufsetzen können, jedoch nicht müssen.

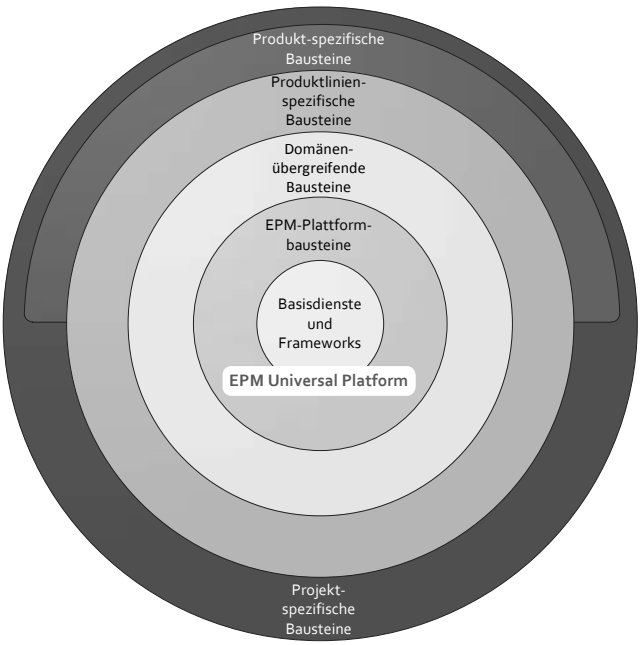


Abbildung 1: Schichtenmodell der EPM-Referenzarchitektur

Für jeden einzelnen Baustein erfolgt in der EPM-Referenzarchitektur eine Definition und Abgrenzung der funktionalen Zuständigkeiten und nichtfunktionalen Eigenschaften, die zur Vermeidung von Überschneidungen mehrerer Bausteine beiträgt und somit die fachliche Interoperabilität sicherstellt. Dies gilt sowohl innerhalb einer Schicht, als auch zwischen den Schichten.

2.3 EPM-Architekturstil

Die Strukturierung der EPM-Referenzarchitektur wird ergänzt durch Schichten-übergreifende und Schichten-spezifische Architekturregeln, die der EPM-Architekturstil festlegt. Diese setzen bewährte Prinzipien in einen Gesamtkontext und entwickeln sie weiter, insbesondere die Ideen von Johannes Siedersleben (Quasar) [Sie04] und Robert C. Martin (SOLID-Prinzipien) [Mar02]. Insgesamt bilden die Regeln den EPM-Architekturstil, der eine von mehreren denkbaren Alternativen zur Realisierung der angestrebten Qualitätseigenschaften darstellt. Die möglichst durchgängige Anwendung dieser Regeln stellen sicher, dass eine technische Interoperabilität sichergestellt ist und Effekte des „architectural mismatch“ [GAO09] vermieden werden. In allen genannten Schichten wird ein konsequent auf Modularisierung und aktivem Management der

Modulabhängigkeiten (physische Softwarestruktur) basierender Software-Entwicklungsansatz verfolgt. In den Schichten (2) bis (5) werden darüber hinaus Komponenten und Schnittstellen (logische Softwarestruktur) definiert, für welche die erste Schicht die gemeinsame Ausführungsinfrastruktur bereitstellt. Im Rahmen dieses Beitrags kann nur ein Überblick über die Architekturstandpunkte (Abschnitt 3) gegeben werden, und anschließend ein Aspekt genauer betrachtet werden, nämlich die Klassifikation von Schnittstellen anhand ihrer Besitzer (Abschnitt 4) und die Erfahrungen mit Standardschnittstellen (Abschnitt 5).

3 Die Architekturstandpunkte des EPM-Architekturstils

Für den EPM-Architekturstil werden vier Architekturstandpunkte im Sinne des ISO-Standards 42010 betrachtet, die unterschiedliche Arten von Softwarebausteine als Modellierungselemente verwenden:

1. Logische Softwarestruktur: Logische Komponenten und Schnittstellen
2. Logische Verteilung: Kompositionskontext und logische Komponenten
3. Physische Softwarestruktur: Module
4. Physische Auslieferung: Deploymenteinheiten, Konfigurationen

Diese werden im Folgenden jeweils kurz beschrieben.

Logische Softwarestruktur: In der logischen Softwarestruktursicht werden *Komponenten* und (Komponenten-)*Schnittstellen* und ihre statischen Abhängigkeiten modelliert. Eine Komponente *implementiert* i.d.R. eine Schnittstelle und *benutzt* eine Menge anderer Schnittstellen. In einem logischen Softwarestrukturmodell kommen keine Abhängigkeiten zwischen Komponenten vor.

Diese Konventionen für die logische Softwarestruktur legen eine gewisse Abbildung auf physische Elemente nahe, zum Beispiel eine auch physische Trennung von Schnittstellen und ihren Implementierungen[3]. Bei der Abbildung bestehen jedoch erhebliche Freiheitsgrade, die grundsätzlich mit beliebigen Technologien umzusetzen sind. Die Granularität von logischen Komponenten und Schnittstellen lässt sich nicht pauschal festlegen, typischerweise ist diese aber gröber als die der Module in der physischen Struktursicht. Insbesondere besteht eine logische Schnittstelle praktisch immer aus vielen Schnittstellen im Sinne einer Programmiersprache.

Logische Verteilung: In der logischen Verteilungssicht können konkrete *Einsatzszenarien* der Komponenten modelliert werden. Hier wird dargestellt, wie Komponenten miteinander zu einer logischen *Konfiguration* verbunden werden, um nach außen einen bestimmten Zweck zu erfüllen. Dabei werden auch logische *Ausführungskontexte* definiert, an deren Grenzen eine physische Verteilung stattfinden kann. Es besteht eine

[3] Zwingend ist dies jedoch nicht, so dass sich auch Legacy-Software auf diese Weise modellieren lässt. Für die Entwicklung neuer Softwarebausteine fordert der EPM-Architekturstil allerdings in der Regel eine solche Trennung.

Wechselbeziehung zwischen der Art der Schnittstellen und den möglichen Einsatzszenarien, da über Ausführungskontexte hinweg keine objektorientierten Schnittstellen, sondern dienstorientierte Schnittstellen eingesetzt werden sollten[4].

Physische Softwarestruktur: In der physischen Softwarestruktursicht sind die Bausteine *Module*, die jeweils implizit oder explizit eine Schnittstelle *exponieren* und die von anderen Modulen exponierten Schnittstellen *nutzen*. Es gibt dabei unterschiedliche *Modultypen*. Eine wichtige Rolle spielen Module, die die logischen Schnittstellen und Komponenten abbilden. Es gibt darüber hinaus aber weitere Modultypen, z.B. Module, die Basisframeworks enthalten, oder solche, die als Adapter zu einem Komponentencontainer dienen. Standardschnittstellen werden generell als eigenständige Module (Schnittstellenmodule) abgebildet, die unabhängig von den Modulen der Komponenten sind, die die Schnittstelle benutzen oder implementieren.

Physische Auslieferung: Die Module werden zu *Auslieferungseinheiten* (z.B. DLLs plus weitere für die Nutzung benötigte Dateien) gebündelt, die für den Betrieb in der Regel durch einen Komponentencontainer instanziert und konfiguriert werden. Die Bündelung soll so erfolgen, dass zur Konfigurationszeit ein Austausch einer Komponente durch eine andere Implementierung derselben Standardschnittstelle bzw. durch eine neue Version derselben Implementierung möglich ist.

4 Standardschnittstellen in der logischen Struktursicht

Schnittstellen können hinsichtlich ihres logischen „Besitzers“[5] (vgl. [Sie04]) unterschieden werden in Importschnittstellen, Exportschnittstellen und Standardschnittstellen. Dies ist in Abbildung 2 dargestellt, die Besitzkontexte sind dort durch die dunklen Kästen markiert. Importschnittstellen werden von einer verwendenden Komponente festgelegt. Exportschnittstellen werden von einer implementierenden Komponente festgelegt. Standardschnittstellen werden unabhängig von verwendenden und implementierenden Komponenten festgelegt.

Exportschnittstellen sind im Allgemeinen der mit Abstand verbreiteteste Fall: Es wird eine Komponente entwickelt, die implizit oder explizit eine Schnittstelle zur Nutzung durch andere Komponenten anbietet. Auch wenn eine Komponenten-orientierte Entwicklung die explizite Trennung von (Export-)Schnittstelle und Komponente fordert, ist damit noch nicht automatisch eine strenge Trennung des Lebenszyklus von Komponente und Schnittstelle verbunden: Wenn bei der Entwicklung der Komponente eine Änderung der Schnittstelle zweckmäßig erscheint, wird diese vorgenommen, ggf. eingeschränkt durch gewisse Zusicherungen über die verlässliche Evolution an die Nutzer. Die Schnittstelle wird aber nicht als eigenständiges Artefakt gepflegt.

Bei Standardschnittstellen ist dies anders. Ihre Evolution wird nicht durch die Entwickler einer einzelnen Implementierung bestimmt, sondern unterliegt einer eigenständigen

[4] vgl. [Sie04] und Martin Fowler’s First Law of Object Distribution: Don’t distribute your objects [Fow03].
[5] Diese Unterscheidung ist nicht zu verwechseln mit der Benutzt- vs. Implementiert-Beziehung zwischen Komponenten und Schnittstellen. Diese Zusammenhänge sind voneinander unabhängig.

Weiterentwicklung. In die Überlegungen zur Weiterentwicklung fließen sowohl Anforderungen von Nutzern und Implementierern ein. Datenbankzugriffsschnittstellen wie ODBC stellen Standardschnittstellen dar, bei der Nutzer, Schnittstelle und Implementierungen sogar typischerweise von drei jeweils unterschiedlichen Organisationen entwickelt werden.

Plug-in-Schnittstellen sind typischerweise als Importschnittstellen ausgeprägt; dies wird beispielsweise in Eclipse von verschiedenen Teilkomponenten verwendet; die Plug-ins sind dann nur in einer bestimmten Komponente nutzbar. Standard-Plug-in-Schnittstellen, die von mehreren Komponenten angeboten werden, sind jedoch ebenfalls denkbar.

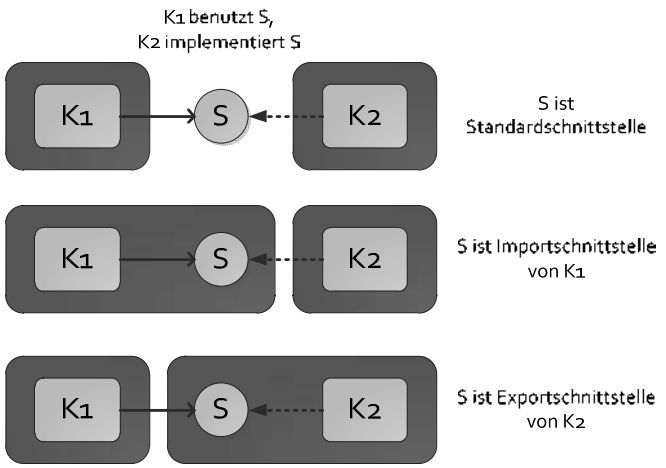


Abbildung 2: Besitzer von Schnittstellen

Für Standardschnittstellen gelten besondere Qualitätsanforderungen (vgl. [Sie04]):

- **Abgeschlossenheit:** Die Schnittstelle muss in sich abgeschlossen sein, d.h. sie darf keine weiteren Schnittstellen verwenden, ansonsten bilden diese lediglich eine logische Schnittstelle, die aus mehreren Teilschnittstellen besteht. Die benötigten Datentypen müssen innerhalb der Schnittstelle definiert werden. Ausnahmen gelten für allgemein nutzbare Bausteine (vgl. [Nag90]) der Schicht (1) Basisdienste und Frameworks (vgl. Abbildung 1).
- **Dokumentation:** Es existiert eine Qualitäts-gesicherte Dokumentation der Schnittstelle und ihrer Operationen.
- **Validierbarkeit:** Es steht eine Testsuite zur Validierung beliebiger Implementierungen zur Verfügung, die definierte Anforderungen an die Testabdeckung erfüllt.
- **Implementierbarkeit:** Mittels mindestens zwei Implementierungen muss demonstriert worden sein, dass die Schnittstelle implementierbar ist. Zu diesen Implementierungen gehören eine „Dummy“-Implementierung und eine echte, produktiv nutzbare Implementierung.

- **Verlässlichkeit:** Eine Standardschnittstelle sollte nur selten geändert werden. Insbesondere sollten Änderungen, die nicht abwärtskompatibel sind, nur sehr selten vorgenommen werden. In solchen Fällen muss für einen definierten Zeitraum die alte Version noch unterstützt werden und Implementierungen der alten Schnittstelle während dieses Zeitraums auch noch gewartet werden.

Neben diesen Anforderungen mit technischen Auswirkungen gelten auch noch Anforderungen hinsichtlich der Veröffentlichung in der EPM-Referenzarchitektur und der Akzeptanz durch den angestrebten Nutzerkreis. Die Erfüllung dieser Qualitätskriterien ist Bedingung für ihre effiziente Nutzung. Eine kritische Rolle dabei spielt die Verlässlichkeit. Voraussetzung dafür ist einerseits, dass die Standardschnittstellen einen Reifungsprozess durchlaufen haben, und andererseits, dass die Nutzungsprofile der Nutzer nicht zu stark divergieren, was durch die eingeschränkte Anwendungsdomäne der nutzenden Produkte und Lösungen innerhalb der Produktlinie erreicht wird.

Eine für die Schicht der EPM-Plattformbausteine (vgl. Abbildung 1) spezifische Regel des EPM-Architekturstils fordert, dass vorrangig Standardschnittstellen definiert werden, zu denen je nach Erfordernis mehrere implementierende Komponenten existieren. Dies kann sowohl zeitlich parallel als auch versetzt gelten: Zeitlich parallel angebotene Implementierungen derselben Schnittstelle können unterschiedliche nicht-funktionale Anforderungen bzw. unterschiedliche technische Rahmenbedingungen erfüllen bzw. verschiedene externe Systeme anbinden. Zeitlich versetzt angebotene Implementierungen derselben Standardschnittstelle können bei einer Weiterentwicklung der Implementierungstechnologie zum Einsatz kommen, die sich auf die Nutzer der Standardschnittstelle nicht auswirkt. In der EPM-Referenzarchitektur wird festgehalten, welche konkreten Standardschnittstellen und Komponenten ihre Implementierungen es gibt bzw. geben soll.

5 Erfahrungen und Herausforderungen

In diesem Abschnitt werden einige Erfahrungen aus der EPM-Produktlinie im Zusammenhang mit Standardschnittstellen und die sich daraus ergebenden Herausforderungen diskutiert, sowie Lösungsansätze skizziert.

Grundsätzlich sind diese Herausforderungen von Standardschnittstellen nicht neu gegenüber der korrekten Verwendung von Schnittstellen im Allgemeinen. Ihre Auswirkungen sind im Zusammenhang mit Standardschnittstellen jedoch weitreichender.

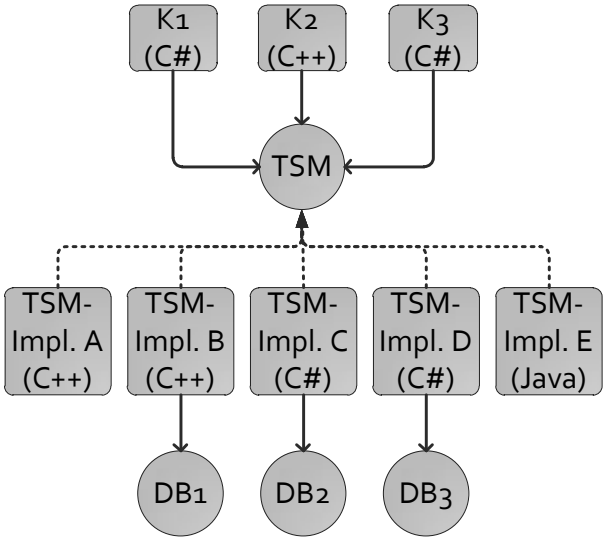


Abbildung 3: Nutzer und Implementierer der TSM-Schnittstelle

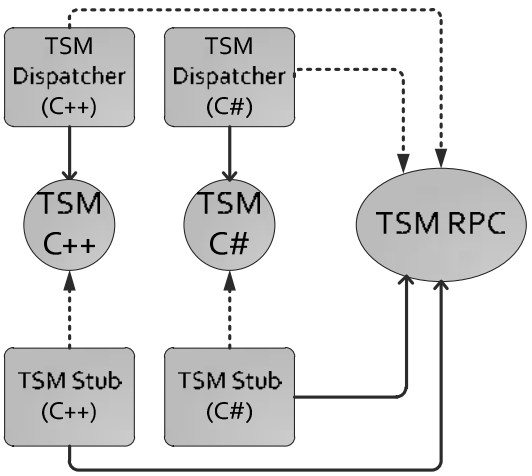


Abbildung 4: Technologie-spezifische Ausprägungen der TSM-Schnittstelle

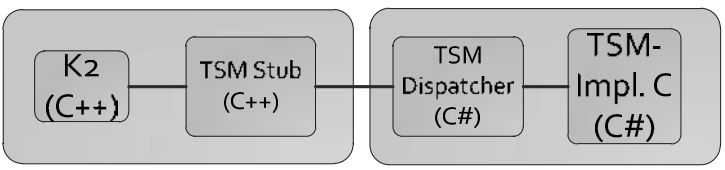


Abbildung 5: Beispiel eines Einsatzszenarios der TSM-Komponenten mit Kompositionskontexten

5.1 Standardschnittstellen in der EPM-Produktlinie: Beispiel Zeitreihen

In der im Aufbau befindlichen EPM-Produktlinie gibt es bereits verschiedene Standardschnittstellen bzw. Vorläufer von Standardschnittstellen mit verschiedenen Implementierungen. Am weitesten gereift ist eine Standardschnittstelle für den Zugriff auf Zeitreihendaten (Time Series Management, kurz: TSM). Es existieren mehrere Komponenten als Implementierungen dieser Schnittstelle, die unterschiedliche Arten der Datenhaltung verwenden (siehe Abbildung 3). Dadurch können sie in unterschiedlichen technischen Umgebungen eingesetzt werden (insbesondere eine Verwendung verschiedener Datenbanken, in der Abbildung als DB1, DB2 und DB3 bezeichnet, alternativ) und weisen unterschiedliche nichtfunktionale Eigenschaften, z.B. hinsichtlich Performance und Datensicherheit, auf. Funktional sind die verschiedenen Implementierungen hingegen grundsätzlich identisch.

Die Schnittstelle existiert in Ausprägungen in C++, C# und Java, und es existieren auch Implementierungen in jeder dieser Technologien. Ein Nutzer der Schnittstelle kann über einen Remote-Zugriffs-Mechanismus die Implementierungen jeder Technologie verwenden (siehe Abbildung 4 und Abbildung 5). In Abbildung 4 ist dies beispielhaft für C++ und C# dargestellt. Je Technologie gibt es einerseits einen sog. Stub, der die Technologie-spezifische Ausprägung der Schnittstelle implementiert, d.h. beispielsweise einem lokalen C++-Nutzer die C++-TSM-Schnittstelle anbietet. Auf der anderen Seite gibt es einen (ebenfalls Technologie-spezifischen) sog. Dispatcher, der eine beliebige konkrete Implementierung seiner Technologie nutzen kann. Stub und Dispatcher kommunizieren über ein Technologie-übergreifend eindeutiges RPC-Protokoll, in diesem Falle das TSM-RPC-Protokoll. Im übertragenen Sinne kann man sagen, dass der Dispatcher diese „Schnittstelle“ implementiert und der Stub diese nutzt. In Abbildung 5 ist eine konkrete Konfiguration (ein Modell der logischen Verteilungssicht) eines C++-Nutzers und einer C#-Implementierung mit dazwischen liegendem Stub und Dispatcher dargestellt. Die Technologie des linken Konfigurationskontexts ist natives C++ mit einem C++-Komponentencontainer, der rechte Konfigurationskontext C#/.NET mit Spring.NET.

5.2 Parametrisierung von Tests gegen Standardschnittstellen

Standardschnittstellen stellen für die Testbarkeit eine besondere Herausforderung dar, bieten aber auch erhebliche Effizienzpotenziale. In einer Testhierarchie bewegen sich diese in der Regel auf einer mittleren Ebene zwischen Systemintegrationstests und modulbezogenen Unit-Tests einer bestimmten Implementierung (bei einfachen Implementierungen, die nur aus einem Modul bestehen, können diese ggf. auch die Rolle von Unit-Tests übernehmen). Eine einmal entwickelte Testsuite kann mit geringem Aufwand für jede Implementierung eingesetzt werden, wenn der Test mit einer Komponente parametrisierbar ist. Dies gilt zumindest dann, wenn die Tests keine impliziten Annahmen über eine bestimmte Implementierung enthalten, und die Implementierungen die Schnittstelle korrekt implementieren, insbesondere das Liskov’sche Substitutionsprinzip [LW94] erfüllen.

5.3 Erweiterte Parametrisierung von Tests mit Testtreibern

Typische Standardschnittstellen können unterschiedlich stark bestimmt ausgeprägt sein (eine vollständige Spezifikation kommt bei den von uns betrachteten Systemen nicht vor). In dem hier betrachteten Fall ist es so, dass es für den Basisteil der Schnittstelle keine funktionalen Unterschiede zwischen den Implementierungen geben sollte. Dies gilt in vielen Fällen aber nicht: Einige Implementierungen akzeptieren bestimmte Parameterwerte, die von anderen nicht akzeptiert werden. Ein typisches Beispiel hierfür ist, dass nur bestimmte Implementierungen einer (Programmiersprach-)Schnittstelle, die zur selben Komponente gehören, akzeptiert werden. Um dieses Problem in den Griff zu bekommen, müssen Tests nicht nur mit der Komponente, sondern mit einem Testtreiber (test subject) parametrisiert werden, der die für den Test benötigten Parameter erzeugen kann.

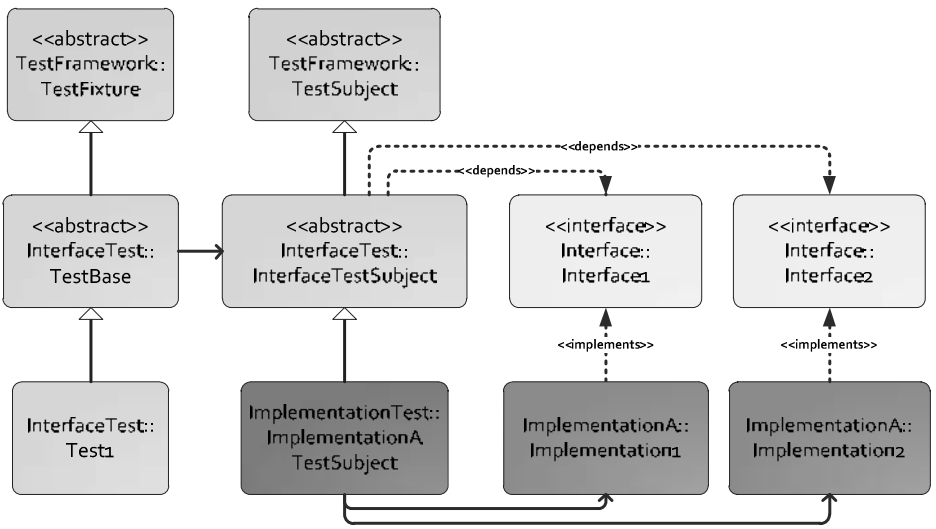


Abbildung 6: Muster für die Trennung von Test und Testtreiber

In Abbildung 6 ist die Klassenstruktur des zugrunde liegenden Entwurfsmusters dargestellt. Das Testframework stellt typischerweise eine abstrakte Basisklasse für Tests `TestFixture` zur Verfügung, ggf. auch eine abstrakte Basisklasse `TestSubject`. `InterfaceTestSubject` ist eine abstrakte Fabrik für die Schnittstelle `Interface1` und `Interface2`; `ImplementationATestSubject` ist eine zugehörige konkrete Fabrik, die von konkreten Implementierung von `Interface1` und `Interface2` abhängt. Es gibt eine abstrakte Testklasse `TestBase` je `InterfaceTestSubject`, die Initialisierung und Deinitialisierung des Testtreibers übernimmt. Dazu kann es mehrere konkrete Testklassen, (in der Abbildung `Test1`) geben, die dasselbe `InterfaceTestSubject` verwenden. Die Verknüpfung von konkretem `TestSubject` und konkretem Test wird erst bei der Instanzierung hergestellt.

5.4 Strukturierung von Standardschnittstellen und zugehörigen Tests

Unterschiede zwischen mehreren Implementierungen einer Schnittstelle, die sich auf die Testbarkeit auswirken, können weitere Ursachen haben: Einerseits kann es sich um unterschiedliche Teilmengen von unterstützten Operationen handeln, die nach Möglichkeit durch mehrere unabhängige physische *Teilschnittstellen* (ggf. auch unabhängige logische Schnittstellen) abgebildet werden können. Dies hat einen weiteren positiven Nebeneffekt: Wie in Abbildung 7 dargestellt, bietet dies dann auch die Möglichkeit, eine generische, aber möglicherweise wenig effiziente Implementierung der erweiterten Schnittstelle nur auf Grundlage der Basis-Schnittstelle anzubieten. Ein Nutzer kann die Implementierung A mit der Schnittstellen-Erweiterung nutzen, obwohl Implementierung A diese selbst nicht implementiert.

Handelt es sich um spezielleres Verhalten derselben Operationen, kann dies durch eine *Schnittstellenhierarchie* abgebildet werden. Für Schnittstellen-basierte Tests ergeben sich Konsequenzen: Im ersten Fall muss der Test mit der Erwartung über die Unterstützung von Teilschnittstellen parametrisiert werden. Im zweiten Fall kommt neben der Schnittstellenhierarchie eine parallele Testhierarchie zum Einsatz. Nur mit der allgemeinsten Testebene ist es unmöglich, aussagefähige Tests mit einer vollständigen Code-Abdeckung zu realisieren.

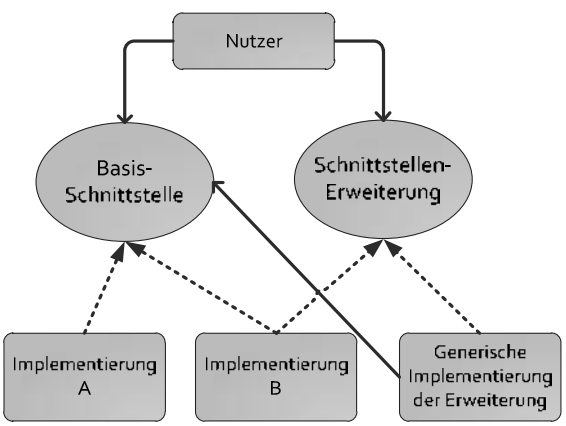


Abbildung 7: Basis-Schnittstelle und Schnittstellen-Erweiterung

5.5 Remote-Fähigkeit von Standardschnittstellen

Eine wichtige Frage ist die Remote-Fähigkeit von Standardschnittstellen. Bei der hier betrachteten Zeitreihenzugriffs-Schnittstelle ist es naheliegend, diese entfernt zu nutzen. Für den gewählten Ansatz der Technologie-übergreifenden Nutzung ist dies sogar zwingend. Eine Remote-Nutzung ist nie transparent möglich: Offensichtlich gilt dies für das Zeitverhalten, aber auch das im programmiersprachlichen Sinne funktionale Verhalten ist betroffen. Die Remote-Fähigkeit ist daher beim Entwurf der Schnittstelle zu berücksichtigen, sie kann nicht ohne weiteres nachträglich hinzugefügt werden, da

dann alle Nutzer angepasst werden müssen. Neben der gröberen Granularität der Operationen und dem Verzicht auf Verteilung auf Objekt-Ebene gilt dies insbesondere für das Fehlerverhalten. Zunächst muss dieses korrekt sein, d.h. es dürfen bei einer Remote-Verwendung nicht zusätzliche, in der Schnittstelle unspezifizierte Ausnahmen erzeugt werden. Es ist ebenfalls nicht hilfreich, in der Schnittstelle pauschal zu spezifizieren, dass generische Ausnahmen erzeugt werden können („throws Exception"), da dies einem Nutzer unmöglich macht, geeignet auf Ausnahmen zu reagieren. Vielmehr ist für die Remote-Nutzung dem Aufrufer die Möglichkeit zu geben, im Falle von Zeitüberschreitungen, Verbindungsabbrüchen und ähnlichem, je nach Bedarf zu reagieren. Hierfür müssen geeignete Konventionen festgelegt werden, so dass ein Nutzer nicht für jede genutzte Standardschnittstelle oder gar jede Operation ein anderes Fehlerbehandlungskonzept implementieren muss.

6 Ausblick

In dem genannten Beispiel hat sich darüber hinaus gezeigt, dass eine sehr hohe Testabdeckung (Implementierung A: 99%) einer Implementierung nicht automatisch auch zu einer ähnlich hohen Testabdeckung anderer Implementierungen führt (Implementierung B: 68%, Implementierung C: 40%). Theoretisch lassen sich hierfür leicht Gründe finden, z.B. die Prüfung von Vorbedingungen in internen Methoden einer Klasse, die durch einen Test der öffentlichen Methoden unmöglich vollständig geprüft werden können. Praktisch stellt sich die Frage, wodurch dies in welchem Ausmaß bei den konkreten Komponenten verursacht wird, und ob sich dieser Effekt ggf. durch geeignete Konventionen gezielt vermindern lässt. Die konsequente Nutzung von Code Contracts (z.B. .NET Code Contracts) könnte hierbei helfen, da hiermit ein definierter Weg zur Spezifikation von Vor- und Nachbedingungen zur Verfügung steht.

In einer weiteren Ausbaustufe der EPM-Produktlinie ist die Etablierung eines einheitlichen Plug-in-Konzepts geplant, um auch die funktionale Variabilität einzelner Produkte effizienter umsetzen zu können.

Literaturverzeichnis

[Fow03] Martin Fowler: Patterns für Enterprise Application-Architekturen, mitp, 2003.

[GAO09] David Garlan, Robert Allen, and John Ockerbloom: Architectural Mismatch: Why Reuse Is Still So Hard, IEEE Software 26(4), July/August 2009, pp. 66–69.

[LW94] Barbara Liskov, Jeannette Wing: A behavioral notion of subtyping, ACM Transactions on Programming Languages and Systems 16(6), November 1994, pp. 1811–1841.

[Mar02] Robert C. Martin: Agile Software Development, Prentice-Hall, 2002.

[Nag90] Manfred Nagl: Softwaretechnik. Methodisches Programmieren im Großen, Springer, 1990.

[Sie04] Johannes Siedersleben: Moderne Softwarearchitektur, Dpunkt Verlag, 2004.

A Formal Method to Identify Deficiencies of Functional Requirements for Product Lines of Embedded Systems

Florian Markert
Computer Systems Group
Technische Universität Darmstadt
markert@rs.tu-darmstadt.de

Sebastian Oster
Real-Time Systems Lab
Technische Universität Darmstadt
oster@es.tu-darmstadt.de

Abstract: Functional requirements that were stated in cooperation with the stakeholders have to be analyzed and reviewed. Deficiencies like incompleteness, contradictions and redundancy within the requirements may lead to an extended development effort. Identifying and resolving these deficiencies in an existing or evolving set of functional requirements for embedded systems is of major importance. Especially, if the requirements describe a set of possible products. Formal methods provide a powerful way to review the requirements automatically. This paper proposes a method adopted from the formal verification of hardware components to uncover the deficiencies within a given set of requirements. The basis of this approach is built by safety properties represented as Linear Temporal Logic (LTL) formulas which are extracted from the requirements. The presented process is evaluated by means of the specification of a car seat.

1 Introduction

Formulating requirements and analyzing them is a vital prerequisite for keeping the development costs low. The set of requirements describing the attributes of a system is called specification. The development process usually starts with a customer delivering a rough specification that is then refined in talks with the contractor. A specification may change, be extended or completed during the development process. This may become an even bigger problem if the requirements describe a set of products, a so called product line (PL). The developer or design team, who must implement an actual product from the PL, relies on a consistent and complete specification. Therefore, single requirements must not contradict each other or become contradictory due to further customer wishes that change the requirements. The customer must define the characteristics and behavior of the desired product carefully and clearly. Specifications need to be complete and must not leave undesired degrees of freedom to the designer. Due to project teams working on single products and their requirements, it may also happen that redundant requirements are added to the set of requirements. Redundant requirements are those that are covered by one or more requirements that are already part of the specification. They do not provide any additional information to the system behavior.

The contribution of this paper is to provide support for requirements analysis of PLs of Embedded Systems by methods adopted from the formal verification of hardware systems and their components. These methods can prove the consistency of a system and deliver a

measure of completeness of the requirements. We put our focus on functional requirements that can be expressed as safety properties expressed in temporal logic. Non-functional requirements are not taken into account because they usually cannot be expressed by means of safety properties. We refer to functional requirements as requirements concerning the observable behavior of an embedded system. They define the valid input/output behavior the user observes when operating the system. The specification must specify the following ingredients with respect to the behavior of the system:

- inputs (sensors, buttons, messages...)
- outputs (actors, motors, messages...)
- temporal-logical relationships between inputs and outputs

If the embedded system is modeled as a black-box with inputs and outputs, the relation between the inputs and outputs is described by the functional requirements given by the specification. They state how an output has to react to a specific input or to a combination of inputs. The reaction of the output may also depend on internal states of the system.

The remainder of the paper is structured as follows: Related work is referenced in Section 2. Our running example is described in Section 3. In Section 4 an overview of the concept is presented. The property-checking algorithm is described in Section 5 and applied to the running-example in Section 6. In Section 7 the results are summarized and an outlook to future work is provided.

2 Related Work

The quality of specifications with respect to the requirements included may vary strongly. Goetz and Rupp show how specifications should be formulated to avoid contradictions and over-specifications [GR03]. Chantree et al. analyze ambiguities in natural language to improve the overall quality of the specification [CNdRW06]. For our approach the quality of the specification is of major importance.

Uncovering inconsistencies in specifications is discussed by Koopman et al. in [KAP08]. Lauenroth and Pohl address the problems that occur when dealing with variability in requirements analysis [LP08]. Both approaches relate to a model that is expressed as an automaton. The automaton must be extracted from the specification and may become very large for complex specifications.

In [HJL96] Heitmeyer et al. present a method to find several classes of inconsistencies within specifications. The SCR-notation (Software Cost Reduction) is used to define the requirements by means of condition, event and mode transition tables. Checking for contradictions or nondeterminism works similar to our approach, but the definition of completeness differs. Heitmeyer et al. state that a system is complete if every input event changes the system state. Our definition of completeness is more restrictive as we state that a system is complete if its outputs are defined for every possible input combination. Therefore,

our check reveals unwanted degrees of freedom within a specification. Furthermore, we normalize the property set which leads to an accurate view on contradictions and incompleteness and allows us to measure the completeness of a specification.

Bormann [Bor09] presents a technique for the analysis of property-sets. This approach lacks of providing a measure of completeness. Furthermore, the properties need to be written as so called transaction properties and cannot be extracted intuitively. Therefore, our approach is based on [OSE07] by Oberkönig et al. which also relates to the completeness of properties. We refer to a functional requirement as a system requirement to the system that states a functional interrelationship of the observable outputs based on the inputs and the current system state. Such an interrelationship is phrased as an LTL property rather than as an automaton.

3 Running Example

As a running example, we use a car seat controller known from the automotive industry. The car seat needs to fulfill the following requirements which we describe in form of an item list:

1. The seat incorporates three motors that can move the seat up/down, left/right and can change the angle of the backrest.
2. Each motor is controlled by a switch that can be brought in the three positions back-stop-front.
3. The motors can only be active up to a car speed of 15 km/h. The seat must not move over that speed due to safety reasons.
4. A seat heater is included into the seat that can reach the states off-on and is also controlled by a switch.
5. The seat heater can only be active if the seat belt is fastened.
6. Only one motor may be activated at a time or the seat heater may be changed.
7. The seat also incorporates an automatic entry function that helps the driver to access the car.
8. When the door is opened the seat moves back and the driver can enter.
9. When the door is closed the seat moves forward until it reaches the initial position.

3.1 Feature Models

Feature models (FM) are frequently used to describe variabilities and commonalities in a PL. A feature model is an acyclic graph consisting of features representing the properties

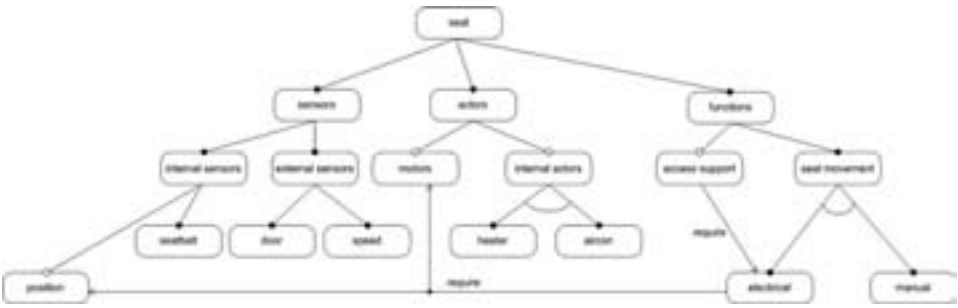


Figure 1: Feature Model of the car seat SPL

of the PL. A Boolean representation of an FM can be obtained according to [CW07]. We need a variability model that can be represented as a Boolean formula in order to add the information provided by the model to the LTL formulas. According to [HST+08] the purposes of a feature model are the following:

- describing common and variable features
- depicting dependencies and constraints between features
- determining which feature combinations are permitted and which are forbidden
- describing all possible products of the SPL

To satisfy these purposes, notations have been defined allowing to describe optional and mandatory features as well as groups of features from which exactly one feature (alternative-group) or at least one feature has to be selected (or-group). The hierarchical as well as cross-tree dependencies such as require and exclude decide which feature combinations are permitted. In [KCH+90] feature models are initially introduced as part of FODA. Fig. 1 depicts the feature model of our running example. The model and the names of the features is one possible representation for the variability incorporated within the PL. The feature model is no canonical representation. There are also other equivalent feature model representations that describe the same PL.

4 Concept

We extract formal properties from the functional requirements. Formal properties provide an unambiguous way to express functional relations between the inputs and outputs of a system. Usually, temporal logics, like Linear Temporal Logic (LTL) [Pnu77], express formal properties. We transform those formal properties into a normal form, and we then check if the properties suffer of one of the following deficiencies:

Contradiction: A contradiction within a set of requirements leads to an inconsistency that cannot be resolved by the system designer. Two or more requirements can contradict each

other by directly stating contradictory behavior, which leads to an over-specified output behavior. Moreover, contradictions may exist that cannot be identified at the first glance due to their indirect dependency on several requirements. A contradiction exists if two possible ways are specified in which the system can behave given the same set of inputs and the same system state.

Incompleteness: The set of requirements may be incomplete. We define that a set of requirements is complete if it describes the output behavior of every actor for every possible combination of inputs. If it is intended that the output behavior is unspecified in certain situations then this don't-care situation can also become part of the resulting property-set.

Redundancy: Two or more requirements are redundant if they state equal behavior of an output. Thus, they are specified in a non-contradictory way. This may trivially happen if several copies of a requirement exist or if a requirement can be deduced from other requirements. If the algorithm identifies these redundant properties, the set of requirements can be reduced without losing information about the system behavior.

Our approach starts with the initial specification, which may be given as plain text only. A state chart, activity diagram, or use-case may also serve as a specification of the system behavior. Functional requirements are extracted from the specification and arranged as a list. The list represents the specification in a semi-formal structure facilitating the extraction of properties. We also build an FM representing the commonalities and variability of the PL. The list of requirements must then be traced to the FM. For every requirement the respective feature or feature combination that causes the requirement to become part of a derived product from the PL must be known. The concept is closely related to the concept presented in [MO10] that aimed at the generation of test oracles from formal properties.

Afterwards, we derive LTL formulas from the previously formatted set of functional requirements according to Section 5. We then add the Boolean representation of the FM to the LTL formulas. The tracing of the features to a respective requirement is done by adding the single features to the LTL formula. The method presented in Section 5 then checks the formal properties automatically. For this check we do not need any further model because the properties incorporate all necessary information about the system behavior. Deficiencies are then detected and given to the user.

As long as deficiencies are found within the property-set, the deficiencies need to be analyzed and resolved in the list of functional requirements. Resolving the deficiencies must be done manually with the help of the results of the checking algorithm. Afterwards, the property-set must be modified due to the changes made within the list of functional requirements. This procedure is repeated until there are no more unwanted deficiencies in the formal properties. As a consequence, the underlying set of functional requirements is free from unwanted deficiencies, too. Since we added the information of the FM to the properties, we can also assure that there are no deficiencies left for every derivable product.

A major advantage of this procedure is that small changes of the functional requirements can be mapped to the existing set of formal properties easily, once the formal properties have been elicited. Using formal properties for requirement representation, newly added, removed, or changed requirements can be checked for adding unwanted deficiencies on the fly. For this procedure no implementation or further model is taken into account. The

properties are checked with respect to themselves. This procedure works with all types of temporal logics which incorporate the expressiveness necessary to describe the behavior of embedded systems. We use LTL since it is an established and well-known temporal logic.

5 Formal Analysis of Requirements

We focus on functional requirements that can be expressed by implications. A functional requirement can describe everything that represents the functionality of a system or a device. Statements like: "When the driver enters, the information to fasten the seatbelt must be displayed" can be seen as a global requirement. It omits the necessary implementation and leaves that to the design team. However, it can also be seen as an implication ($driver_enters \rightarrow information_displayed$).

A requirement that is extracted from a specification is called property in the area of formal hardware verification. Thus, the set of all requirements is called property-set. In order to describe a system formally, the properties may have to be extracted from a non-formal specification. We will explain the procedure by a small example taken from the running example:

...The seat heater may only be activated if the seat belt is fastened...

This excerpt leads to the following implication in LTL:

$$G(\neg seat_belt \rightarrow X(\neg seat_heater)) \tag{1}$$

If the seat belt is not fastened, the seat heater must not be active in the next cycle. As the requirement is modeled as a safety property the LTL formula begins with the Globally operator *G*. The Next operator *X* expresses the temporal relation between the left and right side of the implication.

Since we take a PL into account, a conjunction of the features that trigger the requirement must become part of the LTL formula. All mandatory features may be omitted because they are always part of the PL and, therefore, always true.

$$G(heater \wedge \neg seat_belt \rightarrow X(\neg seat_heater)) \tag{2}$$

This formula is only checked if the seat heater (*heater*) is part of the derived product.

Fig. 2a shows a path which is defined by the left side of an LTL formula. The next state of this path is undefined because none of the requirements defines a value for a certain output on the right side of the implication. Fig. 2b shows a path which has two concurrent next states. Two left sides of implications are valid on the same path but lead to two different next states defining contradictory values for the same output with their right sides. Fig. 2c shows a scenario describing redundancy. A specification consists of three requirements R1, R2 and R3. All the behavior which is defined by R3 has already been defined by R1 and R2. Thus, the requirement R3 is redundant. Considerable attention has been given to the problem of completeness-checking of test suites in the past few years A number of so-called "coverage tests" have been developed which examine different aspects of a test suite. Those tests usually deal with code coverage or control flow analysis [Lig02]. In the

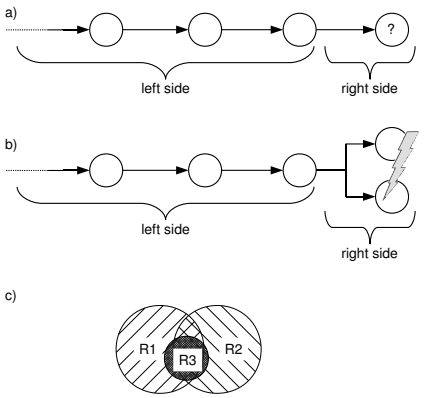


Figure 2: Possible Deficiencies

area of hardware simulation it has been state-of-the-art to use coverage metrics in order to evaluate the completeness of the simulation environment for more than ten years [CK93]. In contrast to that, our approach does not ask the question "Which part of the system is covered" but "Do the requirements incorporate all relevant functionality?".

In the field of formal verification of hardware systems (model checking with formal properties), a procedure was developed that is capable to analyze a specification consisting of properties with respect to its completeness [OSE07]. The completeness analysis calculates the *degree of determination* of all outputs within a specification. Checking the completeness of formal requirements may result in a gap. An output is called fully determined if it is set to a defined value for every combination of inputs and internal states.

We assume that inputs and outputs of embedded systems can only be set to the values '0' and '1' as known from digital hardware. Thus, other data types like integers or complex data types must be first transformed into an equivalent representation using bit vectors. Its size must be sufficient to cover the range of all possible values that the system can assign to the integer value.

Properties are subdivided into two groups: One limits an output in a specified case to '0' and the other one to '1'. The disjunction of all properties' left sides that force the output to become '0' define the so-called off-set (v_0). Accordingly, the disjunction of all properties' left sides forcing the output to become '1' defines the on-set (v_1).

We define the determination function and the consistency function from v_0 and v_1:

$$Determination\ function := v_0 \vee v_1 \tag{3}$$

$$Consistency\ function := v_0 \wedge v_1 \tag{4}$$

If the determination function equals '1' the output is fully determined (complete). In all other cases it is partly or completely undetermined.. Therefore, an output is fully determined if the disjunction of the values forcing it to '0' or '1' represents the complete set of inputs in all combinations. The output is then driven to a specific value on all possible input combinations without leaving any degrees of freedom. The counterpart of the determination

function characterizes all situations in which the output is not determined.

The consistency function must always be equal to '0'. Otherwise, the specification is inconsistent. If the conjunction of (v_0) and (v_1) becomes true, there must be at least one property that forces the respective output to '0' and '1' at the same time. The consistency function defines all situations in which the properties are inconsistent. These situations must be corrected within the requirements.

Definition: If the determination function of an output equals '1', the output is fully determined. In all other cases the following metric defines the percentage of the determined situations:

$$degree\ of\ determination = \frac{\#minterms}{2^n} \tag{5}$$

where n is the number of variables of the determination function and ***#minterms*** corresponds to the number of satisfied assignments in all Boolean variables of the function. A satisfied assignment in all Boolean variables corresponds to a single 1 in a Karnaugh map.

If there are, for instance, the following requirements for the output c given (as Boolean expression)

$$a \rightarrow c$$
$$\neg a \wedge b \rightarrow \neg c$$

the Karnaugh maps in Fig. 3 result. The diagrams represent the off-set, on-set and determination function of c, respectively. The Karnaugh maps depend on the two inputs *a* and *b*. In this example, the off-set is not the inverted on-set because the two properties do not fully determine the output c. Therefore, the degree of determination is 75 % since three minterms of a total of four are specified by the Boolean requirements.

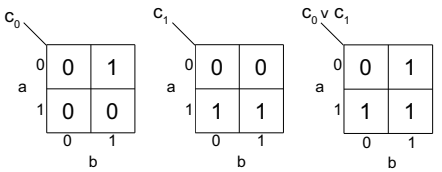


Figure 3: Karnaugh Maps of the Off- and On-Set

If the consistency function does not equal '0', inconsistencies have been found which represent contradictions within the specification. A metric for the degree of consistency is obsolete. For the algorithm that is capable of transforming properties into an off- and on-set please refer to [OSE07].

6 Exemplary Workflow

In the following we will present an exemplary workflow based on the running example from section 3. We will first write a complete and non-contradictory set of LTL properties using the procedure explained in section 4. Afterwards, we will demonstrate how contradictory and incomplete situations are detected.

6.1 Preparation

First of all, we formulate the requirements as LTL properties. Afterwards, we apply the procedure presented in section 5 to normalize the property-set and create microproperties. The algorithm uses the microproperties to analyze the requirements considering consistency and completeness. A degree of determination of 100 % is shown and it is proven formally that the properties are free from contradictions. The following two paragraphs show the main problems of the consistency analysis aided by slightly modified properties. Tab. 1 shows the number of generated microproperties, the total degree of determination, and the runtime. All results were produced on a Core2Duo system with 3 GHz using a single core.

Table 1: Results of the Car Seat Example

experiment	microproperties	determination degree	time
initial properties	40.282	100 %	35 s
with gap	40.274	97 % with 2 gaps	55 s
with contradiction	40.354	100 % with 250 contradicitions	34 s

6.2 Discovering Incompleteness

We modified a property for the seat heater according to equation (6) in order to create an incomplete property set. Equation (6) describes the behavior of the seat heater extracted from the requirements at the first glance. The completeness check detects a specification gap and returns a degree of determination for the seat heater (*Heat_output*) of 50%. The gap is depicted in Fig. 4 as a path. The diagram reveals the non-determined output due to the specification gap and the history leading to its non-determinism. The user is then requested to complete the path by adding the property that defines the output in the next state. In the example the behavior of the seat heater was not specified in the situation when the seat belt is not fastened.

$$G(Seatbelt \wedge Heat_request \rightarrow X(Heat_output)) \tag{6}$$

Therefore, equation (6) needs to be changed as follows:

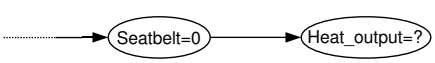


Figure 4: Path Representation of the Specification Gap

$$\begin{aligned} &G((Seatbelt \wedge Heat_request \rightarrow X(Heat_output)) \vee \\ &\quad (\neg Seatbelt \rightarrow X(\neg Heat_output)) \vee \\ &\quad (Seatbelt \wedge \neg Heat_request \rightarrow X(\neg Heat_output))) \end{aligned} \tag{7}$$

Equation (7) defines *Heat_output* in all possible combinations of the inputs *Seatbelt* and *Heat_request* and leaves no further gaps. According to the gap free formal LTL specification

the initial set of requirements can be revised. The functional requirement (5) from section 3 should be written in a clearer way as follows:

- If the seatbelt is not fastened the heater must not be active.
- If the seatbelt is fastened the heater must be active if it receives a request from the corresponding switch and must be inactive otherwise.

These formulations lead to only one possible way in which the seat heater may operate and leave no degrees of freedom to the designer.

6.3 Discovering Contradictions

To force the specification to be contradictory we use the properties in Equation (11) to (14). Equations (11) to (13) describe the behavior of one of the engines with three properties. To ensure the readability of the resulting LTL expressions we substituted parts of the properties with corresponding macros (*M1; M2; M3*).

$$\begin{aligned} M1 :&(Engine2_request = 0) \wedge \\ &(Engine3_request = 0) \wedge (Speed_Sensor < 16) \end{aligned} \tag{8}$$

$$M2 :(Engine1_request = 1) \wedge (Seat_Back > 0) \tag{9}$$

$$M3 :(Engine1_request = 2) \wedge (Seat_Back < 50) \tag{10}$$

$$G((M1 \wedge M2) \rightarrow X(Engine1_output = 1)) \tag{11}$$

$$G((M1 \wedge M3) \rightarrow X(Engine1_output = 2)) \tag{12}$$

$$G((M1 \wedge \neg M2 \wedge \neg M3) \rightarrow X(Engine1_output = 0)) \tag{13}$$

Equation (14) is a small excerpt of the LTL property leading to the contradiction. It defines the output behavior due to bus conflicts of one or more accesses to the bus simultaneously.

$$\begin{aligned} G(((&Heat_request \neq 0) \wedge (Engine1_request \neq 0)) \rightarrow \\ &X((Engine1_output = 0) \wedge (Invalid_Inputs = 1))) \end{aligned} \tag{14}$$

The four properties have reached a size which does not allow the stakeholder to uncover the contradiction at the first glance. The formal analysis discovers a contradiction which can be represented as a path (Fig. 5). The path depicts the contradiction within the specification of the output of the motor by having two contradicting next states. It occurs when two devices try to access the bus simultaneously. Within this example an assumption of the property belonging to the motor was not stated that excludes the seat heater to be active together with the motor.

To resolve the contradiction *M1* needs to be changed as shown in Equation (15).

$$\begin{aligned} M1 :&(Engine2_request = 0) \wedge (Engine3_request = 0) \\ &\wedge (Speed_Sensor < 16) \wedge (Heat_request = 0) \end{aligned} \tag{15}$$

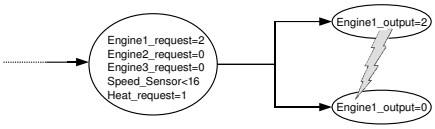


Figure 5: Path Representation of the Contradiction Discovered

Due to this change *Engine1_output* can only be active if no other device is accessing the bus simultaneously. In the initial set of properties the heater was omitted which lead to the contradiction.

Redundant properties can be discovered by leaving them from the property set. If the degree of determination remains unchanged the property is redundant.

7 Conclusion and Future Work

The later the stakeholder identifies deficiencies in a specification or in a set of requirements, the more time and effort is needed to redefine the affected parts of the system. Within the scope of this paper it could be shown that requirements can be tested for incompleteness, contradictions and redundancy with formal methods taken from the hardware verification. The challenge of this approach is to formulate the requirements by means of the property specification language LTL. Our approach is suitable to review specifications which were formulated by a stakeholder. Since the resulting property-sets can easily and intuitively be modified and extended, our approach can be used in the field of requirements engineering. When extending the LTL property set with the boolean representation of the FM, we can also check requirements of PLs. Especially, when reviewing evolving specifications the approach is capable to help maintaining the consistency.

As a next step, we need to review the kind of requirements that can be modeled with property specification languages. Up to now, we used the algorithm exclusively for systems that can be described by a set of safety properties. We also need to determine how errors in the extraction process of properties from the specification impact the completeness and if they can be identified. The maximum problem size that can be handled must also be evaluated.

LTL is used in many different domains to specify system behavior. Therefore, we need to figure out whether our approach can handle specifications from other areas, like business processes. In principle the approach should be suitable for any specification that can be expressed as safety properties.

Finally, we will try to adopt the method presented in [Bor09] to tackle the completeness problem. We need to evaluate the differences with respect to the drawbacks and advantages of both methods.

References

[Bor09] Jörg Bormann. *Vollständige funktionale Verifikation*. PhD thesis, Universität Kaiserslautern, 2009.

[CK93] Kwang Ting Cheng and A. S. Krishnakumar. Automatic Functional Test Generation Using The Extended Finite State Machine Model. In *DAC '93: Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 86–91, New York, NY, USA, 1993. ACM Press.

[CNdRW06] Francis Chantree, Bashar Nuseibeh, Anne de Roeck, and Alistair Willis. Identifying Nocuous Ambiguities in Natural Language Requirements. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 56–65, Washington, DC, USA, 2006. IEEE Computer Society.

[CW07] Krzysztof Czarnecki and Andrzej Wasowski. Feature Diagrams and Logics: There and Back Again. In *Proceedings of the 11th International Software Product Line Conference*, pages 23–34, Washington, DC, USA, 2007. IEEE Computer Society.

[GR03] Rolf Goetz and Chris Rupp. Psychotherapy for System Requirements. In Dilip Patel, Shushma Patel, and Yingxu Wang, editors, *IEEE ICCI*, pages 75–80. IEEE Computer Society, 2003.

[HJL96] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Trans. Softw. Eng. Methodol.*, 5:231–261, 1996.

[HST+08] Patrick Heymans, Pierre-Yves Schobbens, Jean-Christophe Trigaux, Yves Bontemps, Raimundas Matulevicius, and Andreas Classen. Evaluating formal properties of feature diagram languages. volume 2, pages 281–302, 2008.

[KAP08] Pieter Koopman, Peter Achten, and Rinus Plasmeijer. Testing and Validating the Quality of Specifications. In *ICSTW '08: Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pages 41–52, Washington, DC, USA, 2008. IEEE Computer Society.

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[Lig02] Peter Liggesmeyer. *Software-Qualität*. Spektrum Akad. Verlag, Heidelberg [u.a.], 2002.

[LP08] Kim Lauenroth and Klaus Pohl. Dynamic Consistency Checking of Domain Requirements in Product Line Engineering. In *RE '08*, pages 193–202, Washington, DC, USA, 2008. IEEE Computer Society.

[MO10] Florian Markert and Sebastian Oster. Model-Based Generation of Test Oracles for Embedded Software Product Lines. In *Workshop Proceedings of the 14th International Software Product Line Conference*, 2010.

[OSE07] Martin Oberkönig, Martin Schickel, and Hans Eveking. A Quantitative Completeness Analysis for Property-Sets. In *FMCAD '07*, pages 158–161, Washington, DC, USA, 2007. IEEE Computer Society.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. *Symposium on Foundations of Computer Science*, pages 46–57, 1977.



Doktorandensymposium

Online Performance Prediction with Architecture-Level Performance Models

Fabian Brosig
Karlsruhe Institute of Technology, Germany
fabian.brosig@kit.edu

Abstract: Today's enterprise systems based on increasingly complex software architectures often exhibit poor performance and resource efficiency thus having high operating costs. This is due to the inability to predict at run-time the effect of changes in the system environment and adapt the system accordingly. We propose a new performance modeling approach that allows the prediction of performance and system resource utilization online during system operation. We use architecture-level performance models that capture the performance-relevant information of the software architecture, deployment, execution environment and workload. The models will be automatically maintained during operation. To derive performance predictions, we propose a tailorable model solving approach to provide flexibility in view of prediction accuracy and analysis overhead.

1 Introduction

Modern enterprise systems based on the Service-Oriented Architecture (SOA) paradigm have increasingly complex architectures composed of loosely-coupled services that are subjected to time-varying workloads and evolve independently. In such dynamic environments, managing system resources so that end-to-end performance requirements are met while resources are efficiently utilized is a challenge. Due to the inability to keep track of dynamic changes in the system environment and predict their effect, today's SOA systems often exhibit poor performance and resource efficiency thus having high operating costs.

System usage profiles, software deployments or execution environments change over time: Services are subjected to time-varying workloads. New services are deployed or existing services are newly composed. The service-providing components are deployed on distributed, heterogeneous system environments including different middleware configurations and virtualization layers. System administrators are often faced with questions such as: What performance would a new service deployed on the virtualized infrastructure exhibit? What would be the effect of migrating a service from one virtual machine to another? How should the workloads of a new service and existing services be partitioned among the available resources so that performance objectives are met and resources are utilized efficiently? How should the system configuration be adapted to avoid anticipated performance problems arising from growing customer workloads? In order to provide for an efficient resource and performance management honoring service level agreements (SLAs), the ability to predict resource utilization and system performance on-line, i.e.,

continuously during system operation, is essential.

Existing approaches to run-time performance prediction (e.g., [MG00, NKJT09]) are based on predictive performance models such as (layered) queueing networks or queueing petri nets abstracting the system at a high level without taking the software architecture or, e.g., the virtualization layer into account. These approaches are often restricted to homogeneous servers, single workload classes and abstract individual services as black boxes.

Approaches to performance prediction at design and deployment time (e.g., [BKR09, SLC+05, Obj]) are mostly based on architecture-level models that are annotated with descriptions of the system's performance-relevant behavior. The underlying modeling languages account for modeling systems at different levels of abstraction and detail as well as modeling performance-relevant context dependencies between execution environments, software deployments and usage profiles. However, most of them suffer from a significant drawback which render them impractical for use at run-time: Given that they are designed for off-line use, the models are *static*, i.e., do not capture the dynamic aspects of systems. Maintaining them manually during operation is prohibitively expensive [Hel09].

We will provide *dynamic service performance models* that describe performance-relevant system design abstractions and are automatically maintained during operation, thus can capture also the dynamic aspects of modern enterprise system environments, where configuration and deployment changes are common. For instance, new services are deployed on-the-fly, virtual machines are migrated or servers are consolidated during operation. The dynamic service performance models will include all information relevant to predict a service's performance on-the-fly. The models will be tied to their deployment platform in order to keep them up to date. A deployment platform will provide interfaces describing their performance-relevant aspects and have built-in self-monitoring functionality. Following [WFP07], we propose a convergence of performance monitoring, modeling and prediction as interrelated activities.

The following goals will be pursued: i) Development of a meta-model for dynamic service performance models capturing static and dynamic performance-relevant information about the software architecture and deployment, the execution environment and the workload. ii) Development of a methodology to integrate the dynamic service performance models with their deployment platforms. The focus will be on automated model maintenance including model parameter estimation. iii) Development of techniques for solving the dynamic service models. To derive performance predictions from dynamic service models at run-time, the models will be transformed into predictive performance models that are then solved by analytical or simulation techniques. We will use multiple solving techniques to provide flexibility in view of prediction accuracy and analysis overhead.

The proposed thesis is part of the Descartes Research Project [KBHR10] which envisions a new kind of SOA platforms that are self-aware of their performance and resource efficiency. They are self-aware in the sense that they will be aware of changes in their environment and be able to predict the effects of such changes on their performance. They will automatically adapt to ensure that system resources are utilized efficiently and performance requirements are continuously satisfied.

2 Related Work

A survey of model-based performance prediction techniques was published in [BDMIS04]. A number of techniques utilizing a range of different performance models have been proposed including queueing networks (e.g., [MG00]), layered queueing networks (e.g., [WZL06]), queueing Petri nets (e.g., [Kou06]), stochastic process algebras [GHKM05], statistical regression models (e.g., [EFH04]) and learning-based approaches (e.g., [EEM10]). Those models capture the temporal system behavior without taking into account its software architecture and configuration. We refer to them as predictive performance models.

To describe performance-relevant system aspects at a different abstraction level, there are a number of meta-models for describing architecture-level performance models specifying the performance and resource-demanding behavior of software systems [Koz09]. The most prominent meta-models are the UML SPT and MARTE profiles [Obj], both of which are extensions of UML as the de facto standard modeling language for software architectures. The architecture-level performance models are built during system development and are used at design and deployment time to evaluate alternative system designs or predict the system performance for capacity planning purposes. Over the past decade, with the increasing adoption of component-based software engineering [CSSW05], the performance evaluation community has focused on supporting component-based systems. Proposed meta-models include SPE-MM [SLC$^+$05], CSM [PW07], KLAPER [GMS07] and PCM [BKR09]. In the latter, the authors advocate an explicit context model as part of the component specification that captures the dependencies of functional and extra-functional properties on the components connections to other components, the execution environment, the allocated hardware and software resources and the usage profile. A modeling notation is proposed allowing component developers to explicitly specify the influence of parameters on the component resource demands as well as on their usage of external services. Component service specifications describe the service's behavior and control flow in an abstract and parametric manner.

The common goal of these architecture-level modeling efforts is to enable the automated transformation into predictive performance models making it possible to predict the system performance and resource consumption for a fixed workload and configuration scenario. However, the existing architecture-level performance models are normally designed for offline use, assume a static system architecture and do not support modeling dynamic environments [KBHR10]. In a virtualized environment changes are common, e.g., service workloads change over time, new services are deployed, or VMs are migrated between servers. The amount of effort involved in maintaining performance models is prohibitive and therefore in practice such models are rarely used after deployment.

3 Goals and Approach

We aim at establishing service-providing systems that are aware of changes in their environment (e.g., service workload, software deployment and execution environment) and

have the ability to predict the effect of those changes on their performance. We intend to provide a new method for online performance prediction by integrating dynamic service performance models with system execution environments. In this section, the main goals and tasks are discussed.

Meta-model for Dynamic Service Performance Models We will develop a meta-model for dynamic service performance models capturing static and dynamic performance-relevant information about service workload, the service's software architecture and the execution environment. Since we intend to keep meta-model instances continuously up to date during system operation, we refer to the models as *dynamic* service models. The modeling language should make it possible to specify information at different levels of abstraction and granularity, since we will use the models in different scenarios with different goals and constraints ranging from quick performance bounds analysis to accurate performance prediction.

Suitable abstractions for capturing the dynamic aspects of multi-layered execution environments (virtualization, operating system, Java virtual machine, application server) have to be developed. The meta-model should allow to describe the layers (e.g., middleware, virtualization) and their performance-influence explicitly [HKKR09] to facilitate modeling an exchange of individual execution layers.

We will use the PCM [BKR09] as a basis, given that it provides support for explicitly modeling the performance-influencing factors of services, i.e., their implementation, execution environment, workload and external service calls. PCM will be augmented by dynamic model parameters and model entities allowing the specification of dynamic system behavior.

Automated Model Maintenance at Run-time We will develop a methodology to integrate dynamic service models with service execution environments. This will include a technique to extract the models semi-automatically based on monitoring data. In order to keep the dynamic service performance models up to date during system operation, we develop methods to automatically maintain them at run-time. Therefore, we plan to integrate the models with the system, i.e., we will connect the performance-relevant system properties with the model instances.

We develop a platform-neutral interface for communicating which components should be monitored, according to which performance metrics should be measured and how the measurement data needs to be processed. This includes application-level monitoring (e.g., method response times), middleware-level monitoring (e.g., database connections) as well as monitoring at operating systems or virtualization layers (e.g., CPU utilization). Given that we observe the system during operation, techniques reducing the monitoring overhead such as sampling and filtering need to be considered.

In particular, the estimation of service resource demands will be of great importance for our approach. In the literature, there exist already many different resource demand estimation methods differing in their accuracy, their robustness and their applicability. For instance, there are notable differences in the amount and type of measurement data that

is required as method input. We aim at providing a resource demand estimation approach that supports tailoring towards specific constraints like available measurement data or the time horizon. For that reason, we work on a classification scheme which will facilitate evaluating the existing resource demand estimation approaches regarding their benefits and drawbacks.

Automated Performance Prediction To derive performance predictions from dynamic service models at run-time, the models have to be composed and transformed (in an automated manner) into a predictive performance model that is then solved by analytical or simulation techniques. The predictive performance models will be generated by means of model transformations using one or more dynamic service models as input. We intend to use multiple existing model solving techniques to provide flexibility in terms of prediction accuracy and analysis overhead. For instance, for a detailed performance analysis a fine-grained simulation could be conducted, whereas an analytical solver based on simple queueing network models or even a coarse-grained bounds analysis would be more appropriate in cases where a quick response is required.

Tailoring towards required prediction accuracy and timing constraints could be accomplished by selecting the appropriate predictive performance model type and level of model granularity on-the-fly. Similarly, the way the predictive performance model is solved, by analysis or simulation, can be varied. We intend to use existing predictive performance models, existing model-to-model transformations, existing simulations and existing analytical techniques that we plan to assemble to a tailorable service model solving process. The methods will be selected based on their expressiveness, overhead, scalability and tool support.

4 Summary

In order to provide for an efficient resource and performance management of modern service-oriented systems, the ability to predict resource utilization and system performance online during system operation is essential. To enable performance predictions in such environments, we propose to use architecture-level performance models that capture the performance-relevant aspects of both the software architecture and the virtualized, multi-layered execution environment. The models will be automatically maintained at system run-time by integrating them with the deployment platform. For performance predictions, we propose a tailorable model solving approach that provides flexibility in view of prediction accuracy and analysis overhead.

References

[BDMIS04] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Trans. Softw. Eng.*, 30(5), 2004.

[BKR09] S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82, 2009.

[CSSW05] I. Crnkovic, H. Schmidt, J. Stafford, and K. Wallnau. Automated Component-Based Software Engineering. *Journal of Systems and Software*, 74(1), 2005.

[EEM10] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. FUSION: a framework for engineering self-tuning self-adaptive software systems. In *Proc. of FSE'10*. ACM, 2010.

[EFH04] E. Eskenazi, A. Fioukov, and D. Hammer. Performance Prediction for Component Compositions. In *Proc. of CBSE*, 2004.

[GHKM05] S. Gilmore, V. Haenel, L. Kloul, and M. Maidl. Choreographing Security and Performance Analysis for Web Services. In *Proc. of EPEW and WS-FM*, LNCS. 2005.

[GMS07] V. Grassi, R. Mirandola, and A. Sabetta. Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *Journal on Systems and Software*, 80(4), 2007.

[Hel09] J. L. Hellerstein. Engineering autonomic systems. In *Proc. of the 6th Intl. Conf. on Autonomic Computing (ICAC)*, pages 75–76, New York, NY, USA, 2009. ACM.

[HKKR09] M. Hauck, M. Kuperberg, K. Krogmann, and R. Reussner. Modelling Layered Component Execution Environments for Performance Prediction. In *Proc. of CBSE*, 2009.

[KBHR10] S. Kounev, F. Brosig, N. Huber, and R. Reussner. Towards self-aware performance and resource management in modern service-oriented systems. In *Proc. of IEEE Intl Conf. on Services Computing (SCC 2010), USA*. IEEE Computer Society, 2010.

[Kou06] S. Kounev. Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets. *IEEE Trans. Softw. Eng.*, 32(7), July 2006.

[Koz09] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 2009.

[MG00] D. A. Menascé and H. Gomaa. A Method for Design and Performance Modeling of Client/Server Systems. *IEEE Trans. Softw. Eng.*, 26(11), 2000.

[NKJT09] R. Nou, S. Kounev, F. Julia, and J. Torres. Autonomic QoS control in enterprise Grid environments using online simulation. *Journal of Systems and Software*, 82(3), 2009.

[Obj] Object Management Group (OMG). UML SPT, v1.1 (January 2005) and UML MARTE (May 2006).

[PW07] D. Petriu and M. Woodside. An intermediate metamodel with scenarios and resources for generating performance models from UML designs. *Software and Systems Modeling*, 6(2), 2007.

[SLC+05] C. U. Smith, C. M. Llado, V. Cortellessa, A. Di Marco, and L. G. Williams. From UML Models to Software Performance Results: an SPE Process based on XML Interchange Formats. In *In Proc. of Intl. Works. Softw Perf. (WOSP)*, 2005.

[WFP07] M. Woodside, G. Franks, and D. C. Petriu. The Future of Software Performance Engineering. In *FOSE '07: 2007 Future of Software Engineering*, 2007.

[WZL06] M. Woodside, T. Zheng, and M. Litoiu. Service System Resource Management Based on a Tracked Layered Performance Model. In *ICAC'06*, 2006.

A Proposal on Validation of an Agile Architecture-Modelling Process

Zoya Durdik
FZI Research Centre for Information Technology
76131 Karlsruhe, Germany

Abstract: Although considered to be incompatible, a combination of agile methods and architectural modelling could open a new perspective in software development. On the one hand, agile methods offer the flexibility required today. On the other hand, architectural modelling is considered as a pre-requisite for the systematic cross-project reuse and for the resulting increase in software developer productivity. In this paper, I define an approach for the empirical validation of a novel process for agile architectural modelling that drives requirements elicitation through the use of patterns and components. I point out the challenges connected to the process development, its design variants (degrees of freedom) and propose a validation concept.[1]

1 Introduction

While agile methods become more and more widespread in the development community [Bab09], a debate about the place of architectural modelling in agile methods is gaining an increasing attention [ABK10].

Agile methods avoid overhead, i.e., activities and artefacts that do not directly contribute to the goals of the project. This fact can be seen in a strong contrast to architectural modelling [ABK10]. This is because the effort of explicitly modelling the architecture of a software system (SA) usually does not pay off during an initial development. However, it pays off rather in the evolution phase or if a cross-project reuse occurs because of the modelled SA [Gor06]. Exactly here lie the drawbacks of agile methods: cross-project reuse of software artefacts, such as patterns, reference architectures and components are not foreseen, and product-lines are not supported. Furthermore, if the architecture is of any concern at all, it is considered to be fixed throughout the project [Bab09].

The challenge addressed in my work is how to reconcile agile methods with architectural modelling in the software development process. For this purpose a cross-gained concept of an architecture modelling process was proposed by me in [Dur10]. This process is driven by the reuse of patterns and components, and itself drives the requirement elicitation by explicitly asking questions back to the customer. It is based on the agile method Scrum [SB08] and QADA [VTT08] as a concept for architectural modelling.

Process design consists of a set of points where multiple decision options are available at a

[1] This work has been supported by the German Federal Ministry of Education and Research (BMBF), grant No. 01BS0822. The author is thankful for this support.

time. Such multiple design decisions are referred to as degrees of freedom (DoF) defining the design of a process. Decisions on what option to choose shall be taken with a strong focus on applicability of the process in practice. Therefore, one of the major challenges while designing the process is to analyse available DoF and to propose the right and well-balanced decision that can be applied on practice. Such decisions should be based on empirical studies, statistical data and/or experiments. Exactly this is usually not the case in a typical process design, where decisions are then first validated in praxis, what can lead to the expensive and time-consuming mistakes. Another challenge is a reasonable trade-off between keeping the process light-weight, keeping the software product maintainable, and making a cross-project reuse possible. Finally, the fundamental validation of such a process is a great challenge as well.

The contribution of this position paper is the discussion of a) degrees of freedom of the proposed agile architecture-driven modelling and requirement engineering (RE) process, and b) possibilities of a fundamental validation of such a process.

This paper is organised as follows: Section 2 briefly presents the overall concept of the proposed architecture-driven modelling and RE process. Section 3 describes several examples of DoF and their relation to the validation concept of the process. Related work is listed in section 4. Finally, Section 5 provides conclusions and an outlook.

2 Basic concept of the agile architecture-driven modelling an requirements engineering process

The goal of my work is to develop a software development process with agile architectural modelling where architectural modelling drives requirements elicitation through the use of patterns and components. This process shall be in line with agile principles, but support software development starting from RE and considering later maintenance and evolution phases. The cross-grained process has been proposed in [Dur10]. Due to the limited place only the brief description of the concept can be presented here.

The proposed process is iterative and incremental, and is summarized in Fig. 1. Each iteration consists of a set of phases – steps that form a single process increment (see Scrum: Sprint). These steps are: initialization, architectural design and components selection that are accomplished with a goal-oriented requirements elicitation. The process steps are repeated in iterations a) inside of each step, and b) together. Each step consists of a set of sub-steps ot shown in the figure, for more details please refer to the above reference.

During these iterations, the artefacts are created according to the demand of the current process stage, therefore, keeping the process lightweight. The innovation lies in the fact that architectural design and the potential reuse of component candidates are drivers of the requirements elicitation process. Please note that the system architectural design is not intended to be complete after the iteration, only a part of the system is designed during it.

Architectural design and component selection shall be supported correspondingly by a pattern catalogue and a component repository. Such catalogues contain a detailed description,

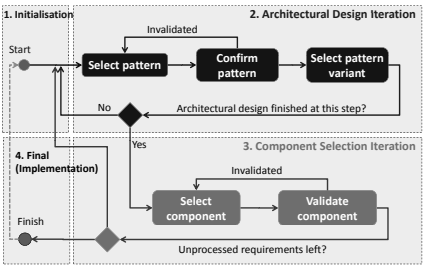


Figure 1: A simplified abstraction of the agile architecture-driven modelling and RE process

i.e. a goal, benefits, drawbacks, application cases, as well as connected or similar patterns or components. It also contains a set of questions that shall help to confirm or invalidate the decision on a pattern or a component and its variant. These questions may help to elicit additional requirements, if the initial information on the system was not sufficient.

This process, however, shall be seen as an example, as its single steps are targets of current and future research. Identified DoF for this steps, i.e. amount of architecture modelling is required, are reviewed in Section 3.

In addition, the benefits of SA need to be exploited in order to support an overall agility. This includes the use of SA for cost estimation, early evaluations of architectural design decisions, and the use of SAs to elicit the right requirements.

3 Degrees of freedom, and concept of the process validation

This section discusses the validation of the proposed process, and exposes its connection to the identified degrees of freedom that are described in detail later on.

Any process usually consists of several steps, and any of these steps can have multiple implementation and execution alternatives. In this paper, such alternatives are referred to as “degrees of freedom” (DoF) of a process design. In the majority of the existing processes, the decisions which variant of a DoF to choose were taken either randomly or based on a personal feeling or the experience of the designer. As the validation of a process is generally a very complicated and labour-intensive task, the possible solution here is to construct a process as a set of “building blocks”, so that each of such blocks can be validated individually. The decisions on each design step and for each DoF shall be taken based on the results of empirical studies, experiments and/or statistics.

Therefore, the refinement of the here proposed process will be executed as a set of smaller steps – building blocks, each explicitly validated. These steps also include evaluation of possible DoF, and validation of design decisions taken in this matter. This will serve as a solid ground for the validation of the complete process assuring the validity of its steps. The validation of a complete process includes validation of the hypothesis that refactoring overhead at the later life-circle stages of a product will be reduced by the explicit architecture modelling supported from the beginning. The validation will involve

2 students groups participating in a controlled experiment. A crossover between the groups shall reduce influence of the learning effect on the experiment results. Nevertheless, a complete and fundamental validation of the process might still not be possible in the frame of my work due to the scale of resources required for it. DoF, which are already identified, will serve as building blocks for the process design. They are listed in the following. Please note that only few DoF could be described here because of the limited space.

DoF 1. How much of the architectural design and architectural modelling is required. Usually the amount of architectural modelling required in a certain project (respectively, project's stage) is determined by the personal experience of the person responsible for the system design (e.g. chief architect or the Team in agile methods). As this research question is not new, and there can be no universal solution, my goal is to analyse currently available solutions and to propose a set of indicators helping to support such decisions.

DoF 2. Decision if the selection of design patterns shall take place before component selection or it shall be done vice versa. Here we have three possible process design decisions. The first option is to introduce the architectural design first, as supposed by the first version of the proposed process. However, the second option is to introduce the component selection first. In this case the design will be built upon the available components. Finally, the third option is to let the person executing the process decide on what is most suitable for the current project. In this case it is important to provide guidelines supporting the selection of the most suitable variant. Such guidelines can be created based on the analysis of a) the statistics on the availability of reusable components (component repositories) in practice, and b) results of an empirical study comparing these two ways.

DoF 3. The fact if architectural decisions are taken at the beginning of the process or are postponed depends on the statistics about to which extend the SA changes during a system's evolution process. If the first SA draft is maintained throughout the development and evolution phases, then a basic SA concept can be defined already during the initial phase of the process. Otherwise, architectural decisions will have to be postponed. For how long such decisions can be postponed is an open research question, as there is no empirical data available to answer it at the moment.

Therefore, an empirical study dedicated to these topic is currently planned. The challenge here lies in the fact that there is no explicit architectural data available for the analysis. Architecture models of open source projects are usually not maintained (or even not available at all). In order to overcome this challenge a reverse engineering tool SoMoX (http://www.somox.org) will be applied to recover the missing models from version to version. So it will be possible to trace if some new features added in the newer versions have produced a significant impact on the SA. It has to be also assured that the changes identified by SoMoX were really introduced into the initial SA plan through new requirements, and are not resulting from the reverse engineering partitioning for components.

Additionally, the results of the study might strongly depend on the domain. This causes another difficulty, as most of the open source projects available for the analysis more or less belong to the category of technical environment applications (e.g. Eclipse or Apache).

Some further examples of the DoF are: decision if SA design and system implementation steps shall be organized as Sprints in Scrum, and a definition of the size of these

steps; decision on form of the SA artefact for the process; and decision if the requirements elicitation shall be frozen during the development phase.

4 Related Work

Generally, most of the articles appearing in this area tend to be rather introductory than practical, and do not propose any exact solution of the problem described above. Several relevant practical approaches are presented in this section.

So [Amb02] provides an introduction into agile methods, UML-modelling and describes general agile modelling methodologies. However the advice given is general, mostly concentrating on the "best practices" of Extreme Programming (XP), and concerned with a very general management level.

The approach presented in [ZBJ04] proposes improving the SA evolution process by mining architecture and design patterns. Although the approach uses a similar idea of information extraction out of patters, its goal is to support the system's evolution and to extract general scenarios for it.

A more practical approach is presented in [Mad10]. It is based on Scrum, XP and sequential project management. Its main goal is SA for communication, quality attributes and technology stack establishment purposes. It does not affect the RE process. Design patterns are used to give a form to implementation, but not to support requirement elicitation and further architectural design. Besides that, the approach seems to be aimed for the similar project types (insurance software systems), as it uses former architectural decisions.

A closely related article [Nus01] states that a SA provides a basis for further requirement discovery and determination of the alternative design solutions. The proposed method is an adoption of the spiral life-cycle model and the author describes it as a complementary to XP. However, the requirements here emerge from: a) the view on models and prototypes b) from the commercially available software (COTS, commercial off-the-shelf software), which shall narrow the possible architectural decisions. The article mentions only the reuse of COTS and requirement identification connected with them. Design patterns are mentioned for "fitting a component-based development approach into the development process". It is a high level article, it neither proposes an exact way how the information from the analysis can flow back from patterns to requirements, nor gives it exact details about the method itself.

Additionally, there is a set of studies on influence of existing SA on the RE, i.e. [MFM09] or [FSH+10], that will be used for a "case study 3" design (Section 3, DoF 3).

5 Conclusions

In this paper, I have discussed a way to validate a novel architecture-driven modelling and RE process proposed by me in [Dur10]. I have listed the major challenges connected to

the process design. I have also described several DoF of such process, pointing out their role in the validation of the whole process.

The future work will have to concentrate on the research of the here presented DoF. According to their type, a required empirical study or an experiment will be designed and conducted to support the selection of the step alternatives of the process, thus assuring that the process steps are based on the experimental / empirical data. In the ideal case, every decision on the process alternative will be validated. This will in no case replace the validation of the proposed process as a whole, but will contribute to the confirmation of the process viability, as well as serve as an additional delta to the already existing processes.

References

[ABK10] P. Abrahamsson, M.A. Babar, and P. Kruchten. Agility and Architecture: Can They Coexist? *Software, IEEE*, 27 , Issue 2:16 – 22, 2010.

[Amb02] S. Ambler. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons, 2002.

[Bab09] M. A. Babar. An exploratory study of architectural practices and challenges in using agile software development approaches. *Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture*, pages 81 – 90, 2009.

[Dur10] Z. Durdik. Architectural Modeling in Agile Methods. *Fifteenth International Workshop on Component-Oriented Programming 2010 at CompArch 2010*, 2010-14:23 –30, 2010. Awarded with CompArch Young Investigator Award.

[FSH+10] R. Ferrari, O. Sudmann, C. Henke, J. Geisler, W. Schafer, and N. H. Madhavji. Requirements and Systems Architecture Interaction in a Prototypical Project: Emerging Results. *16th International Working Conference on Requirements Engineering: Foundation for Software Quality, Lecture Notes in Computer Science*, Volume 6182/2010:23 – 29, 2010.

[Gor06] I. Gorton. *Essential Software Architecture*. Springer, 1 Edition, 2006.

[Mad10] J. Madison. Agile-Architecture Interactions. *Software, IEEE*, 27 , Issue 2:41 – 48, 2010.

[MFM09] J. A. Miller, R. Ferrari, and N. H. Madhavji. Characteristics of New Requirements in the Presence or Absence of an Existing System Architecture. *2009 17th IEEE International Requirements Engineering Conference*, pages 5 – 14, 2009.

[Nus01] B. Nuseibeh. Weaving Together Requirements and Architectures. *Computer*, 34 , Issue 3:115 – 119, 2001.

[SB08] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Pearson Studium, 2008.

[VTT08] Technical Research Centre of Finland VTT. QADA, Quality-driven Architecture Design and quality Analysis. http://virtual.vtt.fi/virtual/proj1/projects/qada/, 2008.

[ZBJ04] L. Zhu, M. A Babar, and R. Jeffery. Mining patterns to support software architecture evaluation. *Fourth Working IEEE/IFIP Conference on the Software Architecture, 2004*, pages 25 – 34, 2004.

Identifikation der Potenziale dreidimensionaler Softwarevisualisierung - Zielstellung und Vorgehen

Richard Müller
Institut für Wirtschaftsinformatik
Universität Leipzig
rmueller@wifa.uni-leipzig.de

Abstract: Die Softwarevisualisierung ist ein wesentlicher Bestandteil der ingenieurmäßigen Softwareentwicklung. Sie macht statische, dynamische und evolutionäre Aspekte von Softwareartefakten in den verschiedenen Phasen des Softwarelebenszyklus sichtbar. Mithilfe von aufgaben- und rollenspezifischen Sichten kann die Software schneller verstanden und ihr Entwicklungsprozess besser gesteuert und kontrolliert werden. Das zentrale Ziel der Arbeit liegt in der Optimierung des Softwareentwicklungsprozesses, indem Nutzenpotenziale der dreidimensionalen Softwarevisualisierung für einzelne Aufgaben dieses Prozesses identifiziert werden. Der tatsächliche Nutzen dieser Potenziale soll dann theoretisch und empirisch überprüft werden und neue Erkenntnisse für Wissenschaft und Praxis liefern.

1 Motivation und Problemstellung

Während des Softwareentwicklungsprozesses werden unterschiedliche Artefakte von verschiedenen Interessensbeteiligten erstellt. Hierzu zählen zum Beispiel von Kunden geäußerte Anforderungen, von Softwarearchitekten erstellte Modelle, von Programmieren geschriebener Quelltext und von Testdurchläufen erzeugte Protokolle. Die Softwarevisualisierung wird eingesetzt, um aufgaben- und rollenspezifische Sichten auf diese Artefakte zu erzeugen, die den Fähigkeiten und Informationsbedürfnissen der betreffenden Interessensbeteiligten entsprechen. Damit eröffnet die Softwarevisualisierung die Möglichkeit, der an sich körperlosen Software ein körperliches Modell zu geben, um dem menschlichen Benutzer die Wahrnehmung, das Verständnis und das Behalten zu erleichtern. Ein korrektes und nachhaltiges Verständnis komplexer Softwaresysteme bildet die Voraussetzung für ein effektives und effizientes Softwareprojektmanagement.

Bisher werden beispielsweise unterschiedliche Arten von UML-Diagrammen (engl. *unified modeling language*) eingesetzt, um Struktur- oder Verhaltensaspekte von Software zweidimensional zu visualisieren. Die einzelnen Diagramme stehen zwar inhaltlich miteinander in Beziehung, doch diese Beziehung selbst ist nicht Bestandteil der Visualisierungen.

Um die Visualisierung eines Softwareartefakts einschließlich der Beziehungen zu anderen Softwareartefakten effizient und ausdrucksstark zu gestalten, bedarf es einer körperlichen

Metapher. Damit werden räumlich sinnvolle Anordnungen der Visualisierungselemente eingegrenzt. Die Metapher kann dann zweidimensional (2D), zweieinhalbdimensional (2,5D) und dreidimensional (3D) dargestellt werden. Dabei ist davon auszugehen, dass entlang der 2D-, der 2,5D- und der 3D-Visualisierung die körperlichen Aspekte immer deutlicher und präziser hervortreten und in virtuellen Umgebungen (engl. *virtual reality environments*) schließlich einer relativ direkten Manipulation zugänglich werden. Dies unterstützt wiederum ihre interaktive Exploration, die dem Menschen im Falle der Interaktion des eigenen Körpers mit anderen Körpern besonders leicht fällt. Demnach ist es erst auf diese Art und Weise möglich, Körper in Raum und Zeit angemessen darzustellen und so eine ganzheitliche Sicht auf Software und ihren Entwicklungsprozess zu vermitteln.

Im Softwarelebenszyklus und in der Softwareentwicklung erwarten die Anwender von Softwarevisualisierungswerkzeugen folgenden projekt- und softwarespezifischen Nutzen [BK01].

- Handhabung der Komplexität von Software
- Erlangung eines besseren Verständnisses von Software
- Kosten- und Zeiteinsparungen bei der Softwareentwicklung und -wartung
- Vereinfachung der Fehlersuche in Software
- Produktivitäts- und Qualitätssteigerungen in der Softwareentwicklung und -wartung

In der Arbeit wird davon ausgegangen, dass sich dieser potenzielle Nutzen der Softwarevisualisierung auf die 3D-Softwarevisualisierung übertragen und weiter steigern lässt. Dazu soll erforscht werden, wie der Softwareentwicklungsprozess durch die 3D-Softwarevisualisierung unterstützt und über die konventionellen Visualisierungsmethoden hinausgehend optimiert werden kann. Der tatsächliche Nutzen der 3D-Softwarevisualisierung im Vergleich zur 2D-Softwarevisualisierung soll dann theoretisch und empirisch untersucht werden. Im Folgenden schließt der Begriff 3D-Softwarevisualisierung auch virtuelle Umgebungen ein.

Primär ist die Arbeit der Softwaretechnik zuzuordnen. Gleichzeitig besitzt sie Bezüge zur Wirtschaftsinformatik, da es darum geht, den Entwicklungsprozess von Informations- und Kommunikationssystemen zu verbessern und in diesem Zusammenhang nicht ausgeschöpfte Nutzenpotenziale aufzuzeigen [FSV05, S. 6 ff.].

2 Stand der Forschung

In der Literatur finden sich verschiedene Definitionen der Softwarevisualisierung. Diese Arbeit folgt der verhältnismäßig weit gefassten Definition von Diehl [Die07, 3 f.], die neben der Visualisierung von Struktur und Verhalten auch die Evolution von Softwareartefakten einbezieht. Softwareartefakte sind alle Zwischen- oder Endergebnisse, die während des Softwareentwicklungsprozesses entstehen. Die Softwarevisualisierung ist auch ein Teilgebiet der Informationsvisualisierung. Dadurch ist es möglich, etablierte Konzepte der Informationsvisualisierung zu adaptieren. Außerdem tangiert sie die Gebiete Mensch-Maschine-Interaktion, kognitive Psychologie und Computergrafik [MCS05].

Eine jede Visualisierung wird in einer Visualisierungspipeline erstellt. Diese spezifiziert eine Prozesskette zur Erzeugung graphischer Repräsentationen [dSB04]. Um eine Darstellung zu erzeugen, müssen Informationen extrahiert, analysiert, gefiltert, auf eine graphische Repräsentation abgebildet und schließlich gerendert werden. Dies gilt gleichermaßen für die Softwarevisualisierung.

Einen effizienten Ansatz, um diesen Visualisierungsprozess umzusetzen, verfolgen Bull et al. mit Model Driven Visualization (MDV) [BSFL06]. MDV ist eine Vorgehensweise zur Erzeugung von Visualisierungen mittels Modellen und Transformationen. MDV baut konzeptionell auf Model Driven Software Development (MDSD) [SVEH07] und Model Driven Architecture (MDA) [Obj03] auf. In [Mül09] wurde dieser Ansatz adaptiert und das modellgetriebene Paradigma mit dem generativen Paradigma (engl. *generative software engineering*, GSE) [CE00] kombiniert, um einen Softwarevisualisierungsgenerator zu entwickeln. Die Erkenntnisse und Ergebnisse aus der vorangegangenen Entwicklung bilden die Basis für diese Arbeit. Mithilfe dieses allgemeinen Visualisierungsprozesses lassen sich sowohl 2D-, 2,5D- und 3D-Visualisierungen erzeugen.

Die Vor- und Nachteile der 2D- bzw. 3D-Visualisierung werden kontrovers diskutiert. Einen kompakten Überblick des anhaltenden wissenschaftlichen Diskurs geben [TC09].

Bei der 3D-Visualisierung werden der höhere Rechen- und Entwicklungsaufwand kritisiert. Der Rechenaufwand spielt aber durch die steigende Rechenleistung eine immer geringere Rolle. Um den Entwicklungsaufwand zu minimieren, wurden unter anderen Modellierungssprachen wie die Virtual Reality Modeling Language (VRML) beziehungsweise deren Nachfolger Extensible 3D (X3D) entwickelt. Eine weitere Schwierigkeit liegt in der Heranführung der Benutzer an neue 3D-Metaphern und Interaktionstechniken. Deshalb sollte bei deren Gestaltung auf Intuitivität und auf Gebrauchstauglichkeit (engl. *usability*) geachtet werden.

Der Mensch lebt und agiert in einer dreidimensionalen Welt. Bei 3D-Darstellungen werden seine visuellen Fähigkeiten ausgenutzt. Ähnelt die 3D-Darstellung darüber hinaus noch der Realität, wird das Verständnis zusätzlich gefördert. Außerdem bietet die 3D-Darstellung einen größeren Raum für Informationen und komplexe Beziehungen können schneller erfasst und verstanden werden. Insbesondere durch die stetige Komplexitätszunahme von Software spielen diese Aspekte eine wichtige Rolle.

Verschiedene Forscher setzen VRML und X3D zur statischen und dynamischen 3D-Softwarevisualisierung ein [ANMB08]. Betrachtet man die Werkzeuge zur 3D-Softwarevisualisierung genauer, lassen sich verschiedene Herausforderungen identifizieren, die bisher nicht hinreichend bewältigt werden konnten [SP91, BK01, TC09].

- Entkopplung der Visualisierungswerkzeuge vom Entwicklungsprozess beziehungsweise der -umgebung
- Dominanz proprietärer Visualisierungsformate
- Geringer Automatisierungsgrad des Visualisierungsprozesses
- Unzureichende Ausnutzung neuer Darstellungs- und Interaktionstechniken
- Fehlende oder unzureichende Untersuchung der Gebrauchstauglichkeit

Neben diesen konkreten Schwächen existierender Werkzeuge zur 3D-Softwarevisualisierung fehlen vor allem auf diesem Gebiet empirische Untersuchungen [TC09]. Aber gerade empirische Untersuchungen sind in der Softwaretechnik und der Softwarevisualisierung erforderlich [HT07, PT07]. Demzufolge müssen bei der Entwicklung von Visualisierungswerkzeugen sowohl die visuelle Repräsentation als auch die dazugehörigen Interaktionstechniken im Kontext einer spezifischen Aufgabe evaluiert werden.

3 Zielstellung und Forschungsfrage

Das Ziel der Arbeit liegt in der Optimierung des Softwareentwicklungsprozesses, indem nicht ausgeschöpfte Nutzenpotenziale der 3D-Softwarevisualisierung für spezifische Aufgaben dieses Prozesses identifiziert werden. Der tatsächliche Nutzen dieser Potenziale soll dann theoretisch und empirisch überprüft werden. Damit wird ein konkreter Forschungsbeitrag zum aktuellen wissenschaftlichen Diskurs 2D versus 3D geliefert. Darüber hinaus sollen die Erkenntnisse und Ergebnisse ebenfalls einen konkreten Mehrwert für die Praxis liefern. Um dieses Ziel erreichen zu können, müssen zwei weitere Teilziele definiert werden.

Einerseits bedarf es eines theoretischen Modells zur Softwarevisualisierung, das den Untersuchungsbereich beschreiben, erklären und prognostizieren kann. Aus diesem Modell werden dann die Hypothesen zur Identifikation der Potenziale abgeleitet.

Andererseits müssen für die empirischen Untersuchungen 2D- und 3D-Visualisierungen einfach und effizient erzeugt werden können. Es wurde bereits dargestellt, dass die Werkzeugunterstützung im Bereich der 3D-Softwarevisualisierung Schwächen aufweist. Deshalb wird ein Softwarevisualisierungsgenerator konzipiert und als Eclipse-Plugin implementiert. Der zu entwickelnde Generator wird in der Lage sein, vollständig automatisiert X3D-basierte 2D- und 3D-Visualisierungen aus Softwareartefakten gemäß Benutzeranforderungen zu erzeugen. Die Grundlage des Generators bildet ein Metamodell zur Softwarevisualisierung. Dieses beschreibt die formale Struktur von Visualisierungen, die mit dem Generator erzeugt werden.

Die Forschungsfrage der Arbeit leitet sich aus dem zentralen Ziel ab und wird durch drei weitere Teilfragen konkretisiert.

- Forschungsfrage: Wo liegen die Potenziale der 3D-Softwarevisualisierung während des Softwareentwicklungsprozesses?
    1. Teilfrage: Welche Aufgaben im Rahmen des Softwareentwicklungsprozesses lassen sich durch die 3D-Visualisierung unterstützen?
    2. Teilfrage: Welche 3D-Visualisierungstechniken eignen sich für welche Aufgaben?
    3. Teilfrage: Wie lässt sich der tatsächliche Nutzen der 3D-Softwarevisualisierung im Vergleich zur 2D-Softwarevisualisierung quantifizieren?

4 Methodisches Vorgehen

Der allgemeine Forschungsrahmen der Arbeit folgt dem konstruktionsorientierten Paradigma [HMPR04]. Im Mittelpunkt stehen die Konstruktion und Evaluation neuer und innovativer Artefakte, die auf die Lösung wichtiger und relevanter Probleme ausgerichtet sind. Das theoretische Modell zur Softwarevisualisierung und der Softwarevisualisierungsgenerator sind solche Artefakte.

Um die zentrale Forschungsfrage zu beantworten, wird dem in Abbildung 1 dargestellten Forschungsprozess gefolgt [Kor07, S. 195, modifiziert]. Dabei werden gezielt Forschungsmethoden der Softwaretechnik und der Wirtschaftsinformatik eingesetzt [PT07, WH07]. Zunächst wird in den relevanten Gebieten nach theoretischen und empirischen Erkenntnissen gesucht, die in Bezug zur Forschungsfrage stehen. Das gesammelte Material wird anschließend metaanalytisch zusammengefasst und ausgewertet. Im nächsten Schritt werden aus dem aggregierten Material Aussagen abgeleitet, um damit das theoretische Modell zur Softwarevisualisierung zu entwerfen. Parallel dazu wird der Softwarevisualisierungsgenerator konzipiert und prototypisch implementiert. Anschließend werden aus dem theoretischen Modell Hypothesen deduktiv und abduktiv abgeleitet. Die Hypothesen werden mit den generierten 2D- und 3D-Visualisierungen in Laborexperimenten empirisch untersucht und überprüft. Hierfür werden etablierte Methoden des Usability-Engineerings (UE) und Evaluationskriterien aus den Taxonomien der Informations- und Softwarevisualisierung miteinander kombiniert. Die Ergebnisse der Hypothesentests bilden dann die Basis zur Klärung der Forschungsfrage respektive zur Identifikation der Potenziale der 3D-Softwarevisualisierung.

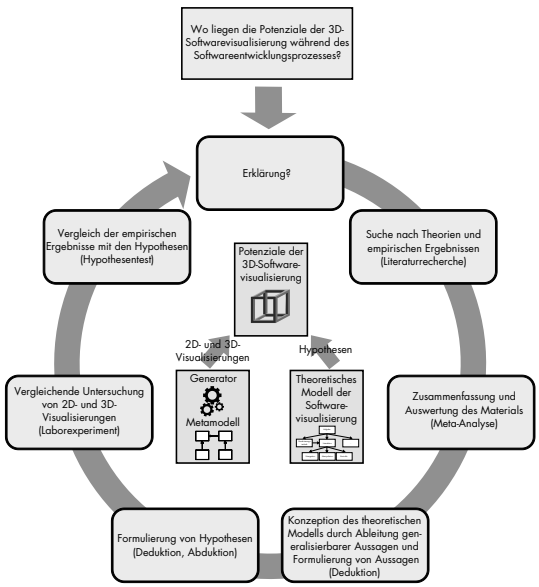


Abbildung 1: Forschungsprozess

Literatur

[ANMB08] C. Anslow, J. Noble, S. Marshall und R. Biddle. Evaluating Extensible 3D Graphics For Use in Software Visualization. In *Symposium on Visual Languages and Human-Centric Computing (VLHCC)*, 2008.

[BK01] S. Bassil und R. K. Keller. Software Visualization Tools: Survey and Analysis. In *Proceedings of the 9th International Workshop on Program Comprehension*, Seiten 7–17, 2001.

[BSFL06] R. I. Bull, M.-A. Storey, J.-M. Favre und M. Litoiu. An Architecture to Support Model Driven Software Visualization. In *International Conference on Program Comprehension*, Seiten 100–106, 2006.

[CE00] K. Czarnecki und U. W. Eisenecker. *Generative Programming Methods, Tools and Applications*. Addison-Wesley, 2000.

[Die07] S. Diehl. *Software Visualization - Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 2007.

[dSB04] S. d. Santos und K. Brodlie. Gaining understanding of multivariate and multidimensional data through visualization. *Computers & Graphics*, 28(3):311–325, Juni 2004.

[FSV05] A. Fink, G. Schneidereit und S. Voß. *Grundlagen der Wirtschaftsinformatik*. Birkhäuser, Mai 2005.

[HMPR04] A. R. Hevner, S. T. March, J. Park und S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[HT07] A. Höfer und W. F. Tichy. Status of Empirical Research in Software Engineering. In *Empirical Software Engineering Issues*, Jgg. 4336/2007, Seiten 10–19. Springer, 2007.

[Kor07] M. Kornmeier. *Wissenschaftstheorie und wissenschaftliches arbeiten: Eine Einführung für Wirtschaftswissenschaftler*. Springer, Dezember 2007.

[MCS05] A. Marcus, D. Comorski und A. Sergeyev. Supporting the Evolution of a Software Visualization Tool Through Usability Studies. In *IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension*, Seiten 307–316. IEEE Computer Society, 2005.

[Mül09] R. Müller. *Konzeption und prototypische Implementierung eines Generators zur Softwarevisualisierung in 3D*. Number 4 in Forschungsberichte des Instituts für Wirtschaftsinformatik. Leipzig, November 2009. ISSN: 1865-3189.

[Obj03] Object Management Group. *MDA Guide Version 1.0.1*. Juni 2003.

[PT07] F. Padberg und W. F. Tichy. Empirische Methodik in der Softwaretechnik im Allgemeinen und bei der Software-Visualisierung im Besonderen. In Gesellschaft für Informatik, Hrsg., *Software Engineering 2007 - Beiträge zu den Workshops*, LNI, Seiten 211–222, 2007.

[SP91] J. T. Stasko und C. Patterson. Understanding and characterizing program visualization systems. Bericht, Oktober 1991.

[SVEH07] T. Stahl, M. Völter, S. Efftinge und A. Haase. *Modellgetriebene Softwareentwicklung - Techniken, Engineering, Management*. dpunkt.verlag, 2007.

[TC09] A. R. Teyseyre und M. R. Campo. An Overview of 3D Software Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(1):87–105, 2009.

[WH07] T. Wilde und T. Hess. Forschungsmethoden der Wirtschaftsinformatik. *WIRTSCHAFTSINFORMATIK*, 49(4):280–287, 2007.

Wertorientierte Portfolio-Optimierung bei Software-Produktlinien

Johannes Müller
Institut für Wirtschaftsinformatik
Universität Leipzig
jmueller@wifa.uni-leipzig.de

Abstract: Software-Produktlinien (SPLs) können durch systematische Wiederverwendung und mittels der Industrialisierung der Softwareentwicklung Wettbewerbsvorteile schaffen. Da die Ausrichtung eines Unternehmens von der Einzelsystementwicklung hin zur Fertigung von Software in SPL eine strategische Entscheidung ist, müssen beim Bestimmen des Umfangs einer SPL neben technischen auch ökonomische Aspekte berücksichtigt werden. In der Arbeit wird ein Prozessmodell entwickelt, das wertbasiert das Portfolio einer SPL optimiert und so ihre Rentabilität erhöht.

1 Einführung

Software-Entwicklung (SE) ist kostenintensiv und geht immer mit einer Verteilung knapper Ressourcen, vor allem der Arbeitszeit der Mitarbeiter, einher. Die Kosten der SE zu kontrollieren, ist ein zentrales Anliegen der SE-Forschung. In den letzten Jahrzehnten wurde die Wiederverwendung von SE-Artefakten als möglicher Weg diskutiert. Neben etlichen gescheiterten Versuchen hat sich die systematische Wiederverwendung mit Hilfe von Software-Produktlinien (SPL) als gangbarer und aussichtsreicher Weg erwiesen [CN07].

Bei SPLs wird Software auf Basis einer gemeinsamen Architektur und gemeinsamer SE-Artefakte entwickelt. Der Softwareentwicklungsprozess umfasst zwei wohlunterschiedene, aber miteinander interagierende Teilprozesse. In der Entwicklung zur Wiederverwendung werden die Software-Produkte (SP) geplant, die involvierten Domänen abgesteckt und die gemeinsame Architektur mitsamt der wiederverwendbaren SE-Artefakte entwickelt und implementiert. Die so gebildete Plattform ermöglicht die Erstellung einer Menge von SP mit gemeinsamen und variablen Merkmalen im zweiten Prozess, der Anwendungsentwicklung. Hier werden ausgehend von den Kundenanforderungen die SP auf Basis der wiederverwendbaren SE-Artefakte erstellt. Die Planung der SP sowie die Analyse der Domänen erfolgt im *Scoping* [Sch03].

In der Software-Branche haben sich die Märkte von Angebots- hin zu Nachfragemärkten entwickelt [BDH08]. Der Erfolg eines Unternehmens hängt zunehmend nicht mehr nur davon ab, gute Produkte anzubieten, vielmehr schaffen erfolgreiche Unternehmen Werte für ihre Kunden, indem sie ihre Bedürfnisse möglichst genau befriedigen [BJ05]. Um erfolgreich zu sein, müssen sie die geschaffenen Werte in Gewinn für sich verwandeln. Der Gewinn wird wesentlich durch den Preis, den Absatz und die Kosten determiniert.

Aufgrund der sehr geringen variablen Kosten von Software-Produkten [BDH08] ist die wertbasierte Preisbildung hier zweckmäßig, die die drei Einflussgrößen auf den Preis, nämlich den Kunden, die Kosten und die Konkurrenz, gleichermaßen berücksichtigt [KC09].

Je mehr Merkmale eine SPL umfasst, umso mehr Bedürfnisse können ihre SP befriedigen und entsprechend viele Kunden ansprechen. Da aber nur begrenzte Ressourcen für die Realisierung zur Verfügung stehen, können nicht alle Merkmale realisiert werden, sondern es muss eine Priorisierung vorgenommen werden. Doch welche Merkmale sollen zuerst realisiert werden? Die, deren Entwicklungskosten sehr gering sind? Oder die, welche den höchsten Kundennutzen stiften? Anscheinend muss ein Anbieter hier abwägen. Zur Unterstützung dieser Entscheidung stehen bisher keine adäquaten Hilfsmittel für Anbieter von SPLs bereit (Abschnitt 4), obwohl diese grundlegenden, immer wiederkehrenden Entscheidungen essentiell für den Erfolg einer SPL sind.

Genau diese Abwägungsentscheidung soll mit den Ergebnissen der Arbeit unterstützt werden. Sie wird ein Vorgehen zur wertbasierten Portfolio-Optimierung (WB-PO), mitsamt den dazugehörigen Methoden und benötigter Software, liefern (Abschnitt 2). Das Produkt-Portfolio definiert die angebotenen Produkte, deren Preis und die anvisierten Kundengruppen. Die WB-PO bei SPLs baut auf der wertbasierten Preisbildung auf.

Die Fragestellung ist für Produktlinien im Allgemeinen relevant. Aufgrund der Besonderheiten von Software erfolgt ihre Untersuchung aber im Kontext von SPL.

Damit die Artefakte einen wissenschaftlichen Beitrag leisten, müssen sie systematisch hergeleitet werden. Die *Design Science* ist ein erprobtes Vorgehen hierfür (Abschnitt 3).

2 Ziele

Im Rahmen der Arbeit wird ein Prozessmodell der WB-PO entwickelt (1), dessen Aktivitäten durch Methoden beschrieben (2) und, falls angebracht, durch Software unterstützt (3) werden.

(1) Ein erster Entwurf des WB-PO ist in Abbildung 1 zu finden. Der Prozess ist als Prozess-

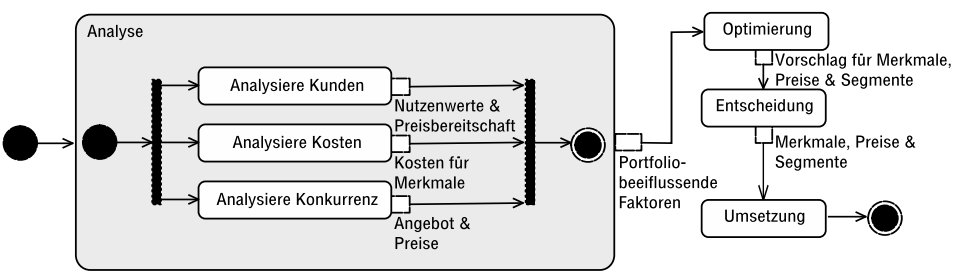


Abbildung 1: Grobgranularer Prozess der WB-PO im *Scoping*

Komponente [OMG08] ausgelegt, der in umfassende *Scoping*-Prozesse, wie etwa PuLSE-Eco [Sch03], integriert werden muss. Er erwartet eine erste grobe Auswahl möglicher Merkmale und liefert Kundensegmente, einen optimierten Merkmalumfang und Preise der Produkte. Die Aktivitäten des Prozesses können verschiedene Methoden realisieren. Deshalb ist der Prozess offen für verschiedene Methoden ausgelegt. Im Rahmen der Arbeit wird einen Satz Erfolg versprechender Methoden entwickelt, angepasst oder ausgewählt.

(2) Grundlage der WB-PO ist die Analyse der Einflussfaktoren auf das Portfolio, nämlich Kunde, Kosten und Konkurrenz. Sie muss zu Beginn erfolgen. Für die Messung des Kundennutzens und der Zahlungsbereitschaft eignen sich Verfahren wie die Conjoint-Analyse [GHH07], das Konkurrenzangebot und deren Preise lassen sich durch Marktanalysen ermitteln [KA06] und die Kosten sind durch Aufwandsmessverfahren, wie etwa dem *Constructive Product Line Investment Model* (COPLIMO) [BBMY04], ermittelbar.

Die Daten über die preisbeeinflussenden Faktoren müssen in geeigneter Weise genutzt werden, um die adressierten Segmente, die angebotenen Merkmale und die verlangten Preise gewinnoptimal zu gestalten. Das resultierende Optimierungsproblem kann in den wenigsten Fällen von einem Menschen überblickt werden. Es muss durch geeignete Verfahren gelöst werden. Im Allgemeinen ist das Problem NP-hart, wenn nicht einige Besonderheiten ausgenutzt werden. Aust schlägt einen Ansatz zur simultanen linearen Optimierung vor, geht aber nicht auf die Besonderheiten von SPL und die Möglichkeiten der Bündelung von Merkmalen zur Gewinnoptimierung ein [Aus96]. In der Arbeit wird Austs Ansatz um die Optimierung von Merkmalbündeln erweitert und die verwendete Kostenfunktion wird an die Besonderheiten von SPL angepasst.

(3) Einige der Aktivitäten des Prozesses können nur mit Softwareunterstützung effizient realisiert werden. Deshalb wird im Rahmen der Arbeit ein Software-Prototyp entwickelt. Dieser wird auf Basis eines Merkmalmodells [CHE04] eine Optimierung des Merkmalumfangs eines Portfolios vornehmen und Preise sowie Marktsegmente ermitteln, die die Rentabilität einer SPL maximieren. Das Merkmalmodell wird um Daten zu den Zahlungsbereitschaften der Kunden, den Kosten der Merkmale und den Preisen der Produkte der Konkurrenz angereichert. Der Software-Prototyp wird die Merkmalkonfiguration und die Preise der Konfigurationen optimieren und für eine gegebene Merkmalkonfiguration den Preis eines Produkts berechnen. Der Prototyp soll auf einem Merkmalmodellierungs-Werkzeug, wie etwa dem *Moskitt Feature Modeler* [Pro10], beruhen und dieses so erweitern, dass die Aktivitäten des Prozessmodells abgebildet werden.

3 Forschungskonzept

Die Basis des Forschungskonzepts bildet die *Design Science* [HMPR04]. Sie verlangt die Konstruktion und Evaluation von Artefakten nach anerkannten Methoden, die bestimmten Kriterien genügen müssen. Dazu gehört, dass die Forschung für einen Problemraum relevante Artefakte hervorbringen und diese im Hinblick auf Nützlichkeit, Effizienz und Qualität evaluieren muss. Dabei leistet sie klar definierte Beiträge, bezieht rigoros bisherige Ergebnisse aus dem Fachgebiet und Nachbargebieten ein und kommuniziert Ergebnisse

so, dass sie sowohl von technisch- als auch von management-orientiertem Publikum nachvollzogen werden können [HMPR04, S. 12 ff.].

Die Entwicklung des Prozessmodells der WB-PO wird unter Verwendung des *Method Engineerings* [Bri96] erfolgen. Die eingesetzten Methoden werden durch eine systematische Literaturrecherche [dABMN+07] identifiziert und an die Anforderungen bei SPL angepasst. So ist beispielsweise angedacht, die Conjoint-Analyse so anzupassen, dass Merkmalmodelle verwendet werden können.

Die Evaluation der Artefakte wird dreistufig erfolgen. Eine theoretische Evaluation soll zeigen, ob Bündelung und Versionierung zur Gewinnsteigerung beitragen können. Ein Experiment soll die prinzipielle Anwendbarkeit des Prozesses und des Prototyps überprüfen. Hier wird das Portfolio einer offen zugänglichen SPL [Uni08] mit WB-PO optimiert und anschließend experimentell über das Internet angeboten. Eine Fallstudie soll ihre praktische Anwendbarkeit demonstrieren.

4 Bisherige Arbeiten

Die Arbeit ist im *Scoping* von SPL verortet, das umfassend erstmalig in [Sch03] als vollständige Methode beschrieben wurde. Danach wird das *Scoping* auf der Ebene des Produkt-Portfolios, der Domäne und der Artefakte durchgeführt, wobei das Produkt-Portfolio als exogen angenommen wird und nur die beiden Ebenen Domäne und Artefakt weiter betrachtet werden. Eine aktuelle Studie zu vorhanden *Scoping*-Methoden identifiziert vier Methoden, die die Ebene Produkt-Portfolio einbeziehen [JE09]. Keine der Methoden verfolgt einen wertorientierten Ansatz. Entweder sie konzentrieren sich auf den Kundennutzen [HHS05; NPB05] oder auf die Kosten [URG10; Tab04]. Die einseitige Konzentration auf Kosten- oder Nutzenaspekte und das Ausschließen des Preises ist aus folgenden Gründen problematisch:

- Deckungsbeitragsgetriebene Priorisierungsentscheidungen benötigen Wissen über mögliche Preise und Kosten.
- Marktsegmente können ohne Wissen über mögliche Preise und Kosten nicht nach ihrer Rentabilität beurteilt werden.
- Ohne Wissen über den Kundenutzen von Produkten einer SPL und den damit möglichen Preisen müssen diese ad-hoc bei jedem Produkt und Kunden gesetzt werden.
- Die Integration von Marketing- und Engineering-Sicht ist nicht möglich.

Diese Feststellung wird auch von Helferich et al. [HSH10, S. 101] vertreten. Sie identifizieren die Preisbildung als einen wesentlichen Baustein für die Merkmalauswahl und die Bewertung der Profitabilität einer SPL, gehen aber nur überblicksartig auf mögliche Methoden und Techniken, wie etwa die Conjoint-Analyse und das *Target Costing*, ein. Dieses Manko bisheriger Methoden wird auch von John und Eisenbarth gesehen, die feststellen, dass die Integration von statistischen Methoden und Optimierungstechniken im *Scoping* von SPLs noch nicht weit fortgeschritten ist [JE09, S. 38].

5 Zusammenfassung

Das Thema WB-PO mitsamt der geplanten Artefakte wurde kurz vorgestellt, seine Wichtigkeit argumentativ und anhand von Quellen belegt und in den bisherigen Forschungsstand eingeordnet. Darüber hinaus wurde dargestellt, wie die Ergebnisse erreicht werden sollen. Damit liegt eine Grundlage für die Diskussion über Relevanz und Ansatz der Dissertation vor.

Literatur

[Aus96] Eberhard Aust. *Simultane Conjointanalyse, Benefitsegmentierung, Produktlinien- und Preisgestaltung*. Number 11 in Entscheidungsunterstützung für ökonomische Probleme. Peter Lang, Frankfurt/Main, 1996.

[BBMY04] Barry Boehm, A. Winsor Brown, Ray Madachy und Ye Yang. A Software Product Line Life Cycle Cost Estimation Model. In *ISESE '04: Proceedings of the 2004 International Symposium on Empirical Software Engineering*, Seiten 156–164, Washington, DC, USA, 2004. IEEE Comp. Soc.

[BDH08] Peter Buxmann, Heiner Diefenbach und Thomas Hess. *Die Softwareindustrie – Ökonomische Prinzipien, Strategien, Perspektiven*. Springer, Berlin, Heidelberg, 2008.

[BJ05] Barry Boehm und Apurva Jain. Developing a theory of value-based software engineering. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, 2005.

[Bri96] Sjaak Brinkkemper. Method Engineering: Engineering of Information Systems Development Methods and Tools. *Information and Software Technology*, 38(4):275–280, 1996.

[CHE04] Krzysztof Czarnecki, Simon Helsen und Ulrich W. Eisenecker. Staged Configuration Using Feature Models. In Robert L. Nord, Hrsg., *SPLC '04: Proceedings of the 9th Software Product Lines Conference*, number 3154 in LNCS, Seiten 266 – 283, Berlin, Heidelberg, 2004. Springer.

[CN07] Paul Clements und Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley Professional, Boston, 6. Auflage, August 2007.

[dABMN$^+$07] Jorge Calmon de Almeida Biolchini, Paula Gomes Mian, Ana Candida Cruz Natali, Tayana Uchôa Conte und Guilherme Horta Travassos. Scientific research ontology to support systematic review in software engineering. *Advanced Engineering Informatics*, 21(2):133–151, 2007.

[GHH07] Anders Gustafsson, Andreas Herrmann und Frank Huber, Hrsg. *Conjoint Measurement: Methods and Applications*. Springer, Berlin u.a., 4. Auflage, 2007.

[HHS05] Andreas Helferich, Georg Herzwurm und Sixten Schockert. QFD-PPP: Product Line Portfolio Planning Using Quality Function Deployment. In J. Henk Obbink und Klaus Pohl, Hrsg., *SPLC '05: Proceedings of the 9th International Software Product Line Conference*, number 3714 in Lecture Notes in Computer Science, Seiten 162–173. Springer, 2005.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park und Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[HSH10] Andreas Helferich, Klaus Schmid und Georg Herzwurm. Product Management for Software Product Lines: An Overview. In Kyo C. Kang, Vijayan Sugumaran und Sooyong Park, Hrsg., *Applied Software Product Line Engineering*. CRC Press, Boca Raton u.a., 2010.

[JE09] Isabel John und Michael Eisenbarth. A decade of scoping: a survey. In *SPLC '09: Proceedings of the 13th International Software Product Line Conference*, Seiten 31–40, Pittsburgh, PA, USA, 2009. SEI.

[KA06] Philip Kotler und Gary Armstrong. *Principles of Marketing*. Pearson, Upper Saddle River, NJ, USA, 11. Auflage, 2006.

[KC09] Hans-Bernd Kittlaus und Peter N. Clough. *Software Product Management and Pricing: Key Success Factors for Software Organizations*. Springer, Berlin [u.a.], 2009.

[NPB05] Elisabeth Niehaus, Klaus Pohl und Günter Böckle. *Software Product Line Engineering: Foundations, Principles and Techniques*, Kapitel Product Management. Springer, Berlin, 2005.

[OMG08] OMG. Software Process Engineering Metamodel, Version 2.0. formal 2008-04-01, Object Management Group (OMG), April 2008.

[Pro10] ProS. MOSKitt Feature Modeler: `http://www.moskitt.org/eng/proyectomfmmoskittfeaturemod/`. Online, 2010.

[Sch03] Klaus Schmid. *Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines*. Fraunhofer IRB, 2003.

[Tab04] Louis J. M. Taborda. Generalized Release Planning for Product Line Architectures. In Robert L. Nord, Hrsg., *Software Product Lines*, Jgg. 3154 of *Lecture Notes in Computer Science*, Seiten 153–155. Springer, Berlin, Heidelberg, 2004.

[Uni08] University of Lancaster. Mobile Media Product Line: `http://sourceforge.net/projects/mobilemedia/`. Online, 2008.

[URG10] Muhammad Irfan Ullah, Günther Ruhe und Vahid Garousi. Decision support for moving from a single product to a product portfolio in evolving software systems. *Journal of Systems and Software*, In Press, Corrected Proof:–, 2010.

Semi-automatische Ableitung externer Anpassungsmechanismen für selbst-adaptive Systeme

Benjamin Nagel

s-lab - Software Quality Lab
Universität Paderborn
Warburger Straße 100
33098 Paderborn
bnagel@s-lab.upb.de

Abstract: Vor dem Hintergrund steigender Komplexität gewinnen selbst-adaptive Systeme immer stärker an Bedeutung. Die Realisierung von Selbst-Adaptivität erfolgt durch Frameworks, die Anpassungslogik in externe Mechanismen auslagern können. Durch Erweiterungen können diese Mechanismen an systemspezifische Anforderungen angepasst werden. Die Ableitung dieser Framework-Erweiterungen aus den Anforderungen wird durch bestehende Ansätze nur unzureichend unterstützt. Selbst-Adaptivität wird in Anforderungsbeschreibungen nur implizit spezifiziert und fehlende Automatismen für die Ableitung führen zu einer erhöhten Fehleranfälligkeit. In diesem Beitrag wird ein Ansatz vorgestellt, durch den externe Anpassungsmechanismen semi-automatisch aus Anforderungen abgeleitet werden können.

1 Einleitung und Motivation

Die steigende Komplexität heutiger Softwaresysteme führt zu immer höheren Anforderungen an die Anpassbarkeit und Wartbarkeit dieser Systeme. Um diesen Anforderungen gerecht zu werden, steht die Softwaretechnik vor der Herausforderung Systeme zu entwickeln, die sich selbstständig an verändernde Bedingungen anpassen können. Diese Systeme werden als „selbst-adaptive Systeme“ bezeichnet.

Die Realisierung selbst-adaptiver Systeme kann durch interne oder externe Ansätze erfolgen. Bei internen Ansätzen wird die Anpassungslogik vollständig in die Software integriert und ist somit direkt mit der Anwendungslogik verknüpft. Die enge Verknüpfung erschwert die Wartung und Wiederverwendung der Anpassungslogik. Aktuelle Ansätze wie das Autonomic Computing [KC03] lagern die Anpassungslogik in externe Mechanismen aus. Durch die Explizitheit der Selbst-Adaptivität werden die Erweiterbarkeit und Wartbarkeit verbessert. Frameworks wie RAINBOW [GCS03] und StarMX [AST09] bieten die technische Grundlage für die Realisierung externer Anpassungsmechanismen.

Während die technische Realisierung selbst-adaptiver Systeme durch bestehende Ansätze unterstützt wird, fehlt eine Methodik zur Ableitung externer Anpassungsmechanismen aus den Anforderungen an ein selbst-adaptives System. Die Anpassungslogik beinhaltet nicht nur komplexe Verhaltensbeschreibungen sondern auch Abhängigkeiten zu System- und Kontextinformationen die innerhalb des Entwicklungsprozesses spezifiziert und realisiert werden müssen.

Das in diesem Beitrag vorgestellte Promotionsvorhaben hat das Ziel, einen Ansatz für die semi-automatische Ableitung von externen Anpassungsmechanismen aus den Anforderungen selbst-adaptiver Systeme zu entwickeln. Hierzu werden im folgenden Kapitel die Problemstellung und die relevanten Forschungsfragen des Promotionsvorhabens beschrieben. Kapitel 3 zeigt bestehende Ansätze in Bezug auf die definierte Problemstellung auf. Der Lösungsansatz für die gegebene Problemstellung wird in Kapitel 4 mit dem erwarteten Beitrag dargestellt. Abschließend wird in Kapitel 5 eine Zusammenfassung gegeben.

2 Problembeschreibung

Das Ziel der semi-automatischen Ableitung ist es, mit möglichst geringem manuellem Aufwand einen externen Anpassungsmechanismus zu generieren, der die Anforderungen an die Selbst-Adaptivität des Systems erfüllt. Bestehende Frameworks bieten eine technische Infrastruktur für externe Mechanismen, die durch Erweiterungen an die systemspezifischen Anforderungen angepasst werden können.

Der prinzipielle Ablauf einer Ableitung ist in Abbildung 1 anhand eines konkreten Beispiels dargestellt. In der Analyse werden Ziele, Anforderungen und mögliche Risikofaktoren identifiziert und spezifiziert. Die Anforderungen können durch kontrolliert natürlich-sprachliche Ansätze wie RELAX[1] [WSB09] mit einer Flexibilität gegenüber ihrer Erfüllung versehen werden. Die Selbst-Adaptivität wird in dieser Phase des Entwicklungsprozesses lediglich implizit durch mögliche Alternativen beschrieben.

In dem dargestellten Beispiel werden die Alternativen „textuelle Darstellung“ und „multimediale Darstellung“ und allgemeine Zielstellungen „möglichst performant“ und „möglichst oft multimediale Darstellung“ angegeben. Als Risikofaktor wird eine „niedrige Bandbreite“ identifiziert, die eine Anpassung der Darstellung erforderlich machen kann. Eine detaillierte Spezifikation der erforderlichen Logik über die Auswahl der verschiedenen Alternativen, wird durch bestehende Beschreibungssprachen nicht unterstützt.

[1] Die Konzepte von RELAX wurden in diesem Beispiel auf die deutsche Sprache übertragen.

Im Design werden Framework-Erweiterungen entwickelt, die die Anpassungslogik in formalen, plattformspezifischen Sprachen beschreiben und so von dem jeweiligen Framework automatisch verarbeitet werden können. Exemplarisch wird in Abbildung 1 eine Anpassungstaktik aus [Ch08] dargestellt, die das Umschalten zur textuellen Inhaltsdarstellung beschreibt. Die Spezifikation der Anpassungslogik erfolgt hier in einer plattformspezifischen Sprache und auf einem sehr detaillierten Modellierungsgrad.

Aus der Spezifikation von Selbst-Adaptivität in den Anforderungen und dem Design ergeben sich zwei wesentliche Problemstellungen für die Ableitung der Anpassungslogik. Die fehlende Mächtigkeit bestehender Beschreibungssprachen erschwert die Spezifikation der geforderten Selbst-Adaptivität auf einer plattformunabhängigen Ebene (Modellierungs-GAP). Außerdem fehlt ein methodisches Vorgehen, um aus den Ergebnissen der Analyse und Anforderungserhebung, die Anpassungslogik abzuleiten (Methodisches GAP).

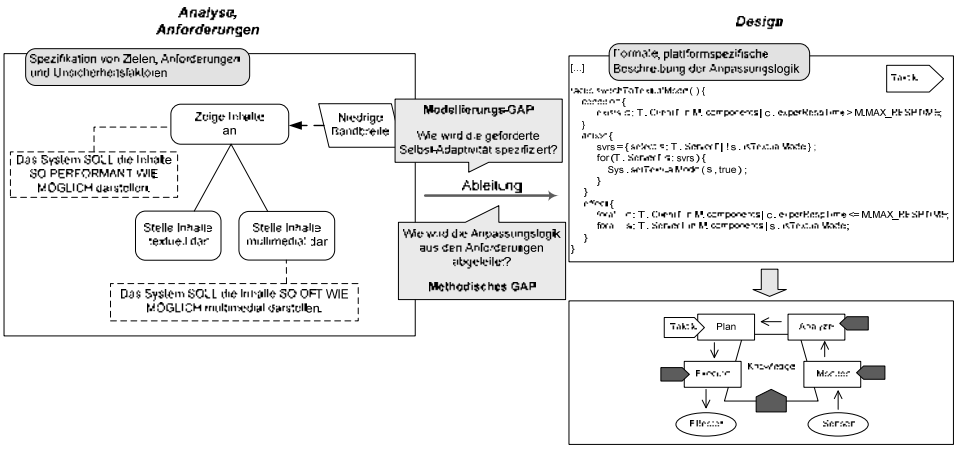


Abbildung 1: Ableitung externer Anpassungsmechanismen

Basierend auf der gegebenen Problembeschreibung, wird für das Promotionsvorhaben die folgende Forschungshypothese aufgestellt: *„Framework-Erweiterungen für externe Anpassungsmechanismen können semi-automatisch aus den Anforderungen selbst-adaptiver Systeme abgeleitet werden.“*. In dem folgenden Kapitel werden bestehende Arbeiten zu der dargestellten Problemstellung aufgeführt und ihre Relevanz für das Promotionsvorhaben aufgezeigt.

3 Verwandte Arbeiten

In der Literatur werden verschiedene Ansätze für die Spezifikation von Anforderungen selbst-adaptiver Systeme aufgezeigt. In [CSB09] werden KAOS Zielbäume [DLF93] mit natürlich-sprachlichen Konzepten von RELAX [WSB09] kombiniert, um Unsicherheiten in der Erreichung von Zielen zu modellieren. Unter Verwendung der Tropos Methodik [BPG04] berücksichtigt [MPP08] auch Abhängigkeiten der Anforderungen zu Elementen in dem System und dem Kontext. In beiden Ansätzen gibt es keine Konzepte, um Selbst-Adaptivität auf einer detaillierten Ebene zu spezifizieren.

In [La03] werden allgemeine, pattern-basierte Ansätze zur Ableitung von Softwarespezifikationen aus Systemzielen beschrieben. Auf Basis von Tropos wurde in [YLL08] eine erweiterte Bibliothek von Pattern für die Ableitung von selbst-adaptivem Verhalten eingeführt. Hierbei werden komponentenbasierte Architekturen abgeleitet, die für eine framework-basierte Realisierung von selbst-adaptiven Systemen zu grobgranular sind. Ein weiterer Nachteil dieser Ansätze ist die fehlende Separierung der Anforderungen an die Selbst-Adaptivität. [ZC06] beschreibt die modellbasierte Ableitung von Petri-Netzen. Dieser Ansatz beschränkt sich auf die Ableitung von Verhaltensaspekten selbst-adaptiver Systeme.

Ein durchgehender Ableitungsansatz von der Anforderungsspezifikation zu konkreten Framework-Erweiterungen, vergleichbar mit dem beschriebenen Vorhaben, ist nicht bekannt.

4 Lösungskonzept

Die Problemstellung und bestehende Ansätze wurden in den Kapiteln 2 und 3 beschrieben. Aufbauend auf den dargestellten Forschungsfragen soll in diesem Kapitel ein Lösungskonzept für die aufgestellte Forschungshypothese skizziert werden. Hierzu wird zunächst der Lösungsvorschlag mit dem erwarteten Beitrag vorgestellt und abschließend die geplante Evaluation des vorgestellten Lösungsansatzes beschrieben.

Die in Kapitel 2 erarbeiteten Forschungsfragen werden durch bestehende Arbeiten in zwei wesentlichen Punkten unzureichend unterstützt:

1.) Die explizite und detaillierte Spezifikation von Selbst-Adaptivität im Entwicklungsprozess selbst-adaptiver Systeme.

2.) Ein automatisiertes Vorgehen zur Ableitung von Framework-Erweiterungen aus der Anforderungsspezifikation.

Für die Adressierung dieser beiden Punkte, wird in dem Promotionsvorhaben der in Abbildung 2 dargestellte, modellbasierte Ableitungsprozess in Anlehnung an [OMG01] vorgeschlagen.

Für die plattformunabhängige Spezifikation von Selbst-Adaptivität wurde die Modellierungsmethodik *Adapt Cases* entwickelt. Dieser use-case-basierte Ansatz ermöglicht die präzise Spezifikation von Selbstadaptivität auf verschiedenen Abstraktionsebenen. Analog zu „traditionellen" Use-Cases ermöglichen die *Adapt Cases* eine schrittweise Ableitung der Anpassungslogik aus den spezifizierten Zielen und Anforderungen. Neben der semi-formalen Spezifikation von *Adapt Cases* in der Use-Case Notation, wurde ein Metamodell zur formalen Spezifikation entwickelt, durch das der Kontext, ein Monitoring-Konzept und die Anpassungsaktionen spezifiziert werden können. Als Grundlage für dieses Modell wird das MAPE-K [KC03] Konzept referenziert. Die Semantik der formalisierten *Adapt Cases* wird durch OCL beschrieben. Durch diesen Ansatz wird eine plattformunabhängige Modellierung der Selbst-Adaptivität ermöglicht.

Die formalisierten Modelle der *Adapt Cases* werden in ein plattformspezifisches Modell für ein bestimmtes Framework, z.B. RAINBOW [GCS03] transformiert. Aus diesem Modell können die einzelnen Erweiterungen für das entsprechende Framework durch eine Modell-Text Transformation generiert werden.

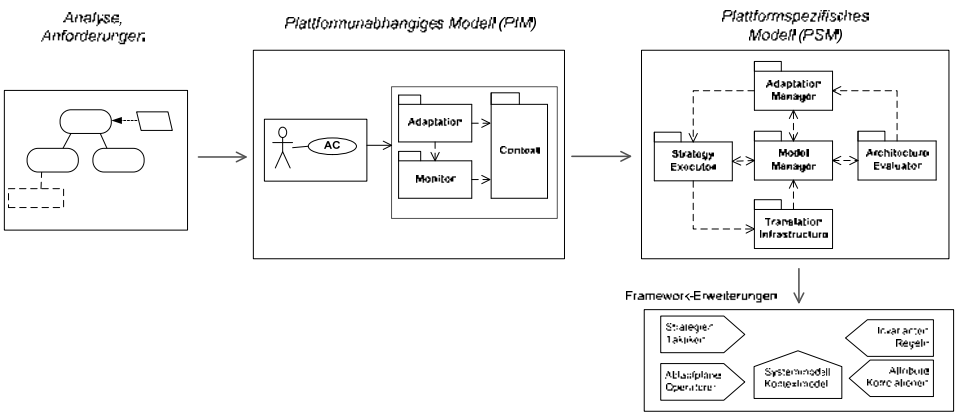


Abbildung 2: Modellbasierter Prozess zur semi-automatische Ableitung

Der vorgestellte Lösungsansatz soll die in Kapitel 2 beschriebenen Forschungsfragen adressieren. Die einzelnen Ableitungsschritte können durch entsprechende Transformationsregeln automatisiert durchgeführt und so der manuelle Anteil minimiert werden. Hierdurch soll eine wesentliche Reduktion von Fehlern im Ableitungsprozess erreicht werden. Um die Anwendbarkeit des Ansatzes in Softwarenprojekten zu ermöglichen, sollen die vorgestellten Konzepte durch ein geeignetes Modellierungswerkzeug unterstützt werden.

Der vorgestellte Lösungsansatz wird im Rahmen des Promotionsvorhabens anhand einer umfangreichen Fallstudie im industrienahen Umfeld des Software Quality Labs (s-lab) evaluiert. Hierdurch sollen die wissenschaftlichen Konzepte validiert und die Anwendbarkeit der Methodik in Entwicklungsprojekten aufgezeigt werden.

5 Zusammenfassung

In diesem Beitrag wurde der Einsatz externer Anpassungsmechanismen für selbst-adaptive Systeme motiviert und Probleme bei der Ableitung dieser Mechanismen aufgezeigt. Es wurde ein konzeptioneller Lösungsansatz aufgezeigt und der erwartete Beitrag in Bezug auf die Problemstellung diskutiert. Als Ausblick wurde die geplante Evaluation im industrienahen Kontext vorgestellt.

Literaturverzeichnis

[AST09] Reza Asadollahi, Mazeiar Salehie, and Ladan Tahvildari. StarMX: A framework for developing self-managing Java-based systems. *Software Engineering for Adaptive and Self-Managing Systems, International Workshop on*, 0:58–67, 2009.

[BPG04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.

[CSB09] Betty Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *Model Driven Engineering Languages and Systems*, volume 5795 of *Lecture Notes in Computer Science*, pages 468–483. Springer Berlin / Heidelberg, 2009.

[Ch08] Shang-Wen Cheng. *Rainbow: cost-effective software architecture-based self-adaptation*. PhD thesis, Carnegie Mellon University, 2008.

[DLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, April 1993.

[GCS03] David Garlan, Shang-Wen Cheng, and Bradley Schmerl. Increasing system dependability through architecture-based self-repair. In *Architecting dependable systems*, pages 61–89. Springer-Verlag, 2003.

[KC03] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[MPP08] Mirko Morandini, Loris Penserini, and Anna Perini. Towards goal-oriented development of self-adaptive systems. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 9–16, Leipzig, Germany, 2008.

[OMG01] OMG. Model Driven Architecture, 2001. http://www.omg.org/mda/.

[La03] Axel van Lamsweerde. From system goals to software architecture. In *Formal Methods for Software Architectures*, pages 25–43. Springer-Verlag, 2003.

[WSB09] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty H.C. Cheng, and Jean-Michel Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, RE*, pages 79–88, Washington, DC, USA, 2009.

[YLL08] Yijun Yu, Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, and Julio Leite. From Goals to High-Variability software design. In *Foundations of Intelligent Systems*, pages 1–16. Springer-Verlag, 2008.

[ZC06] Ji Zhang and Betty H.C. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of the 28th international conference on Software engineering*, pages 371–380, Shanghai, China, 2006.

Model Synchronization for Mechatronic Systems

Jan Rieke*

Heinz Nixdorf Institute & Department of Computer Science
University of Paderborn, Paderborn, Germany
jrieke@upb.de

Abstract: The development of complex mechatronic systems requires the close collaboration of different disciplines on models of the system under development. During such a model-based development, inconsistencies between the different discipline-specific models are likely to occur, which could lead to increased development time and costs if they remain unresolved. Model transformation and synchronization are promising techniques to detect and resolve such inconsistencies. However, existing model synchronization solutions are not yet powerful enough to support such an application scenario. My goal is to extend and improve model synchronization techniques so that they allow for synchronized models with multiple views and abstraction levels, as required by this development process.

1 Introduction and Problem Description

From home appliances to transportation systems, modern mechatronic systems become more and more complex and incorporate an increasing amount of software. This increasing complexity poses challenges to the development of such advanced mechatronic systems.

The design guidelines for mechatronic systems, like the VDI 2206 [Ver04], or the development methods elaborated in the Collaborative Research Center (CRC) 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" in Paderborn, propose that experts from all disciplines collaborate in a first development phase, called the *conceptual design*. Especially, they work out the so called *principle solution*, a system model that is supposed to capture all interdisciplinary concerns, but does usually not contain information relevant only to one discipline. However, in practice, the principle solution rarely captures all interdisciplinary concerns. Additionally, changes to the overall system design may become necessary later, e.g., due to changing requirements. Therefore, cross-disciplinary changes may become necessary during the discipline-specific *refinement* phase.

As an example, consider the construction of a vehicle. The systems engineer attaches a new sensor, because without it the vehicle is unsafe. This change must be propagated to the software engineer, because the sensor's data needs to be processed. Otherwise, the vehicle would not break in time, causing the risk of crashing into a preceding vehicle.

*supported by the International Graduate School Dynamic Intelligent Systems.

Model transformation and synchronization techniques are a promising approach for such synchronization scenarios. Languages and algorithms for bidirectional model synchronization are an intensively researched topic today. However, existing model synchronization techniques mainly focus on basic application scenarios where models of the same or similar expressiveness have to be kept consistent. If, like in our case, models of different abstraction levels, different scopes, or of different domains have to be synchronized, these techniques are often insufficient. Similar issues arise for many model-based development processes (e.g., MDA), too. My goal is to improve existing model synchronization techniques to be able to support such advanced requirements.

I use Triple Graph Grammars (TGG) [Sch94], a declarative transformation language similar to the OMG QVT-R standard [GK10], because we made good experiences with it during the last years and have good tool support for it. TGGs can be applied bidirectionally and allow an intuitive, graphical transformation rule specification.

2 Example and Challenges

Fig. 1 shows an exemplary process, where a cross-disciplinary change occurs. The principle solution is transformed into the different discipline-specific models (1.). The engineers from the different disciplines now start refining their models. E.g., the electrical engineers model the power supply and distribution (2.). Then, the software engineer runs a hazard analysis [GT06] and detects a flaw in the system design: A single distance sensor is not sufficiently reliable, so that a safe distance cannot be guaranteed. Therefore, he proposes to add another sensor to increase the reliability (3.). This change is relevant to other disciplines: E.g., the mechanical engineer has to attach the sensor to the chassis, and the electrical engineer has to connect it to the power supply. Thus, the principle solution has to be updated, because it should reflect all discipline-spanning concerns (4a.). Finally,

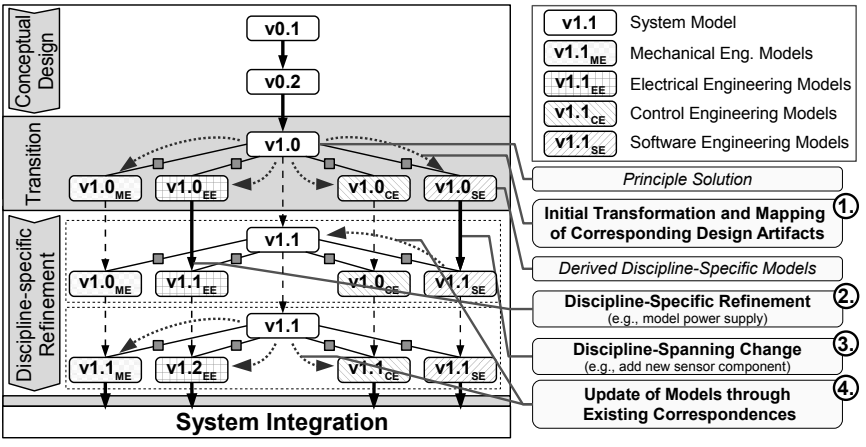


Figure 1: Possible development process with model transformation and synchronization

this updated principle solution is used to update to other discipline-specific models (4b.), preserving manual changes that occurred in the meantime.

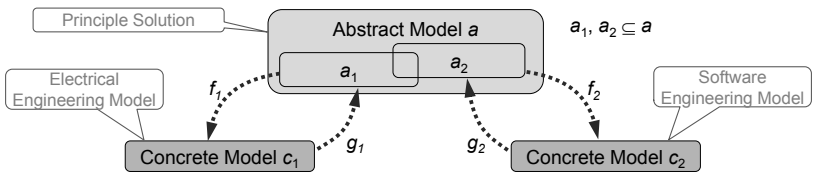


Figure 2: Abstract view of the synchronization scenario

Fig. 2 shows an abstract view of the synchronization scenario described above. There is one abstract model a, the principle solution. Only some parts of a are relevant to a specific discipline model. E.g., the part a_1 is relevant to the concrete model c_1, where a_2 is relevant to c_2. The different parts of a are typically not disjoint, as there are shared elements between the different disciplines. In the course of the development, all of these models may be subject to change. This leads to several challenges in the context of model synchronization, which will be described in the following.

2.1 Definition of abstraction and concretization functions

The abstract model a is used to initially generate the concrete models c_i. E.g., its part a_1 is transformed into c_1 by the transformation function f_1. As a_1 is more abstract, there is a semantic gap between this source and the target c_1 of the model transformation f_1. Usually, the model transformation concretizes all elements of a_1 using defaults. However, the engineers may decide not to stick to this default, but instead use another possible concretization (or refine the default one). Hence, more formally speaking, f_i is not single concretization function, but a family of functions: one abstract model a_i maps to several concrete models $c_{i,j}$. Therefore, the abstraction functions g_i can not be bijective.

Today's model transformation and synchronization techniques lack explicit support for the specification of such families of transformation functions (or non-bijective transformation functions) which are required for synchronizing models on different abstraction levels. Therefore, it is often impossible or at least difficult and non-intuitive to model different concretization rules and store these concretization decisions.

TGGs do not allow for such abstraction relations, either. Usually, TGGs only provide a single rule that translates a specific model element. As we would like to define a family of transformation functions, a first approach would be to specify additional, alternative rules for the same element. However, having different rules for the same element causes unpredictable transformation results in most existing transformation tools, because the order of the rule application is not deterministic: The transformation tool simply selects one of the rules and does not check alternatives.

Therefore, the idea is that, when applying a rule, we should also check for alternative rules that might be applicable. These could then be selected manually by the user where nec-

essary. A critical pair analysis on the rule set could help identifying ambiguous situations (and hiding confluent alternatives). However, in our scenario, transformations should often run automatically, which means that (predefined) default concretizations should be used. Afterwards, the user should be able to search for alternative concretizations on selected elements, or even specify new concretizations which then could be used on other elements, too. Thus, the rule set itself is subject to changes during the development.

2.2 Incrementally updating without changing unaffected target model parts

Both the abstract model a and the concrete models c_i contain parts that are not subject to every transformation. For example, the implementation details of a software component are only contained in c_2, because they are not relevant to other disciplines. In a, only the part a_1 is covered by the transformation rules when translating it to c_1. Therefore, the models cannot be synchronized by running the whole transformation from scratch, because such model-specific information would be lost. Instead, the idea is to *incrementally update* the models by revoking invalid rule applications and applying new rules.

If a change occurs in a, we need to check whether this change has to be propagated to the c_i models. If, for example, an element of c_1 is altered so that the applied rule is invalid, existing model synchronization tools would update a_1 by deleting the corresponding elements (rule revocation), and then try to apply new rules. As there could be elements in a that referenced the deleted elements, this deletion could result in dangling links, i.e., an invalid model. Thus, such simple incremental updates are insufficient for our application scenario.

The idea is to propagate the *editing operations* that took place on the source model. However, specifying a complete transformation on editing operations is a complex task, as there might be a large number of possible editing operations, some of which cannot be anticipated during rule design. Thus, my idea is to emulate such editing operations by applying traditional rules more "intelligent". For example, this can be achieved by not deleting model elements on rule revocation right away. Instead, they are *marked for removal* and could later be reused (possibly with minor modifications), as elements created by new rule application are often "equal" to some deleted elements. The challenge here is to identify which elements to modify or reuse.

2.3 Concurrent modifications, conflict detection and resolution

In a distributed development environment, several engineers from different disciplines work on their models independently. As it is well-known in software development with source code, this can lead to conflicts. Some of these conflicts can be resolved automatically, but user interaction is still necessary in many cases.

The problem even enlarges when working with interconnected models: The conflict could be due to a change in another discipline. In such a case, manual conflict resolution is

ideally done by (a) the systems engineer using the principle solution, or (b) the discipline expert using his own models. As the conflict affects several models, its resolution has to be consistently applied to all models. Thus, the results from a model differencing and merging tool should be included in the model synchronization process.

3 Related Work

Due to lack of space, I focus on the most related work. There is more related work, e.g., on model merging and on non-graph-based approaches.

Giese and Wagner [GW09] present an approach how TGGs can be applied for incrementally updating a target model when changes to a source model occurred, basically by revoking (deleting) and re-applying rules. Giese and Hildebrand [GH09] improved this algorithm so that it removes less elements, but this is still insufficient for my application scenario, because their algorithm still causes unnecessary deletions for more complex editing operations, leading to possible damage of model-specific information.

Ráth et al. [RVV09] propose a solution which does not define the transformation between models any more, but maps between model manipulation operations. This is a fundamentally different approach, as the rule design differs significantly from traditional, declarative model transformations, and its design methodologies still have to be elaborated. Furthermore, designing a complete transformation based upon editing operation is more extensive than using traditional declarative rules. However, I would like to investigate how their ideas could be included in my approach, e.g., to avoid user interaction ambiguous cases.

In the context of chemical process modeling, Körtgen [Kör09] developed a synchronization tool for the case of a simultaneous evolution of both (source and target) models. She defines several kinds of damage types that may occur and gives abstract repair rules for these cases. At runtime, these general repair rules are used to derive concrete repair operations for a specific case. In this way, changes can be propagated with repair operations that delete less elements, i.e., affect less elements that are not subject to the transformations. The approach also includes means for processing alternative rules. However, the solution relies on user-interaction during the transformation, which should, in my application scenario, be avoided where possible.

Xiong et al. [XSHT09] present a synchronization technique that also allows for the simultaneous evolution of both models in parallel. Basically, they run a backward transformation into a new source model, and then use model merging techniques to create the updated final source model. The same is done in forward direction. This technique mainly relies on the capabilities of the model merger. Furthermore, if the model transformation damages model-specific elements during update, their technique cannot avoid information loss, neither. Additionally, they do not incorporate advanced support for conflict detection and resolution, a major requirement in practical scenarios.

4 Summary

Model synchronization is promising technique to support the model-based development of mechatronic systems by automating consistency management tasks, which are time-consuming and error-prone if done manually. However, existing model synchronization techniques mainly focus on simple application scenarios where models of the same or similar expressiveness have to be kept consistent. If models of different abstraction levels or of different scopes have to be synchronized, these techniques are often insufficient.

My aim is to improve existing model synchronization techniques to be able to support such advanced requirements. My hypothesis is that such advanced techniques could greatly improve the development of mechatronic systems as well as other model-based development approaches. I plan to provide a prototype and to apply the developed techniques in the context of a larger example from the CRC 614, in order to evaluate the benefits and possible disadvantages.

References

[GH09] Holger Giese and Stephan Hildebrandt. Efficient Model Synchronization of Large-Scale Models. Technical Report 28, Hasso Plattner Institute at the University of Potsdam, 2009.

[GK10] J. Greenyer and E. Kindler. Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars. *Software and Systems Modeling (SoSyM)*, 9(1), 2010.

[GT06] H. Giese and M. Tichy. Component-Based Hazard Analysis: Optimal Designs, Product Lines, and Online-Reconfiguration. In *Proc. of the 25th Int. Conference on Computer Safety, Security and Reliability (SAFECOMP), Gdansk, Poland*, pages 156–169, 2006.

[GW09] Holger Giese and Robert Wagner. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling*, 8(1), 2009.

[Kör09] Anne-Thérèse Körtgen. *Modellierung und Realisierung von Konsistenzsicherungswerkzeugen für simultane Dokumentenentwicklung*. PhD thesis, RWTH Aachen University, 2009.

[RVV09] I. Ráth, G. Varró, and D. Varró. Change-driven model transformations. In *Proc. of Model Driven Engineering Languages and Systems*. Springer, 2009.

[Sch94] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In G. Tinhofer, editor, *20th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG'94)*, volume 903 of *Lecture Notes in Computer Science (LNCS)*, Heidelberg, 1994. Springer Verlag.

[Ver04] Verein Deutscher Ingenieure. Design Methodology for Mechatronic Systems, 2004.

[XSHT09] Yingfei Xiong, Hui Song, Zhenjiang Hu, and Masato Takeichi. Supporting Parallel Updates with Bidirectional Model Transformations. In *Proc. of the 2nd Int. Conference on Theory and Practice of Model Transformations (ICMT '09)*. Springer-Verlag, 2009.

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Language to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Busi-ness 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3.Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3.Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Erfahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometrics and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für inforrmatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): “Heute schon das Morgen sehen“

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT: Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik – Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der Anwendungsentwick-lung und -wartung im Fachbereich Wirtschaftsinformatik

P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics

P-141 Peter Loos, Markus Nüttgens, Klaus Turowski, Dirk Werth (Hrsg.)
Modellierung betrieblicher Informations-systeme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management

P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung

P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)

P-145 Knut Hinkelmann, Holger Wache (Eds.)
WM2009: 5th Conference on Professional Knowledge Management

P-146 Markus Bick, Martin Breunig, Hagen Höpfner (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Entwicklung, Implementierung und Anwendung
4. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2009)

P-147 Witold Abramowicz, Leszek Maciaszek, Ryszard Kowalczyk, Andreas Speck (Eds.)
Business Process, Services Computing and Intelligent Service Management
BPSC 2009 · ISM 2009 · YRW-MBP 2009

P-148 Christian Erfurth, Gerald Eichler, Volkmar Schau (Eds.)
9th International Conference on Innovative Internet Community Systems
I^2CS 2009

P-149 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
2. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-150 Jürgen Münch, Peter Liggesmeyer (Hrsg.)
Software Engineering
2009 - Workshopband

P-151 Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)
PRIMIUM
Process Innovation for Enterprise Software

P-152 Jan Mendling, Stefanie Rinderle-Ma, Werner Esswein (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 3rd Int'l Workshop EMISA 2009

P-153 Andreas Schwill, Nicolas Apostolopoulos (Hrsg.)
Lernen im Digitalen Zeitalter
DeLFI 2009 – Die 7. E-Learning Fachtagung Informatik

P-154 Stefan Fischer, Erik Maehle Rüdiger Reischuk (Hrsg.)
INFORMATIK 2009
Im Focus das Leben

P-155 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2009:
Biometrics and Electronic Signatures
Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-156 Bernhard Koerber (Hrsg.)
Zukunft braucht Herkunft
25 Jahre »INFOS – Informatik und Schule«

P-157 Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, Peter Stadler (Eds.)
German Conference on Bioinformatics 2009

P-158 W. Claupein, L. Theuvsen, A. Kämpf, M. Morgenstern (Hrsg.)
Precision Agriculture Reloaded – Informationsgestützte Landwirtschaft

P-159 Gregor Engels, Markus Luckey, Wilhelm Schäfer (Hrsg.)
Software Engineering 2010

P-160 Gregor Engels, Markus Luckey, Alexander Pretschner, Ralf Reussner (Hrsg.)
Software Engineering 2010 – Workshopband
(inkl. Doktorandensymposium)

P-161 Gregor Engels, Dimitris Karagiannis Heinrich C. Mayr (Hrsg.)
Modellierung 2010

P-162 Maria A. Wimmer, Uwe Brinkhoff, Siegfried Kaiser, Dagmar Lück-Schneider, Erich Schweighofer, Andreas Wiebe (Hrsg.)
Vernetzte IT für einen effektiven Staat
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2010

P-163 Markus Bick, Stefan Eulgem, Elgar Fleisch, J. Felix Hampe, Birgitta König-Ries, Franz Lehner, Key Pousttchi, Kai Rannenberg (Hrsg.)
Mobile und Ubiquitäre Informationssysteme
Technologien, Anwendungen und Dienste zur Unterstützung von mobiler Kollaboration

P-164 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2010: Biometrics and Electronic Signatures Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-165 Gerald Eichler, Peter Kropf, Ulrike Lechner, Phayung Meesad, Herwig Unger (Eds.)
10th International Conference on Innovative Internet Community Systems (I^2CS) – Jubilee Edition 2010 –

P-166 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
3. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-167 Robert Krimmer, Rüdiger Grimm (Eds.)
4th International Conference on Electronic Voting 2010
co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-168 Ira Diethelm, Christina Dörge, Claudia Hildebrandt, Carsten Schulte (Hrsg.)
Didaktik der Informatik
Möglichkeiten empirischer Forschungsmethoden und Perspektiven der Fachdidaktik

P-169 Michael Kerres, Nadine Ojstersek Ulrik Schroeder, Ulrich Hoppe (Hrsg.)
DeLFI 2010 - 8. Tagung der Fachgruppe E-Learning der Gesellschaft für Informatik e.V.

P-170 Felix C. Freiling (Hrsg.)
Sicherheit 2010
Sicherheit, Schutz und Zuverlässigkeit

P-171 Werner Esswein, Klaus Turowski, Martin Juhrisch (Hrsg.)
Modellierung betrieblicher Informationssysteme (MobIS 2010)
Modellgestütztes Management

P-172 Stefan Klink, Agnes Koschmider Marco Mevius, Andreas Oberweis (Hrsg.)
EMISA 2010
Einflussfaktoren auf die Entwicklung flexibler, integrierter Informationssysteme
Beiträge des Workshops der GI-Fachgruppe EMISA (Entwicklungsmethoden für Informationssysteme und deren Anwendung)

P-173 Dietmar Schomburg, Andreas Grote (Eds.)
German Conference on Bioinformatics 2010

P-174 Arslan Brömme, Torsten Eymann, Detlef Hühnlein, Heiko Roßnagel, Paul Schmücker (Hrsg.)
perspeGKtive 2010
Workshop „Innovative und sichere Informationstechnologie für das Gesundheitswesen von morgen“

P-175 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 1

P-176 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 2

P-177 Witold Abramowicz, Rainer Alt, Klaus-Peter Fähnrich, Bogdan Franczyk, Leszek A. Maciaszek (Eds.)
INFORMATIK 2010
Business Process and Service Science – Proceedings of ISSS and BPSC

P-178 Wolfram Pietsch, Benedikt Krams (Hrsg.)
Vom Projekt zum Produkt
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschafts-informatik (WI-MAW), Aachen, 2010

P-179 Stefan Gruner, Bernhard Rumpe (Eds.)
FM+AM`2010
Second International Workshop on Formal Methods and Agile Methods

P-180 Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, Holger Schwarz (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW)
14. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)

P-181 Michael Clasen, Otto Schätzel, Brigitte Theuvsen (Hrsg.)
Qualität und Effizienz durch informationsgestützte Landwirtschaft, Fokus: Moderne Weinwirtschaft

P-182 Ronald Maier (Hrsg.)
6th Conference on Professional Knowledge Management
From Knowledge to Action

P-183 Ralf Reussner, Matthias Grund, Andreas Oberweis, Walter Tichy (Hrsg.)
Software Engineering 2011
Fachtagung des GI-Fachbereichs Softwaretechnik

P-184 Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.)
Software Engineering 2011
Workshopband
(inkl. Doktorandensymposium)

P-185 Hagen Höpfner, Günther Specht, Thomas Ritz, Christian Bunse (Hrsg.) MMS 2011: Mobile und ubiquitäre Informationssysteme Proceedings zur 6. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2011)

The titles can be purchased at:

Köllen Druck + Verlag GmbH
Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de